UNIVERSITY OF OSLO
Department of Informatics

Full custom implementation of a high performance input buffered switch architecture

Joar Martin Østby

Dr Scient thesis

Research Report 242

ISBN 82-7368-158-0
ISSN 0806-3036

April 1997

Acknowledgements

This is my thesis for the Doctor Scientiarum degree. The main part of this work has been carried out at the University of Oslo, Department of Informatics, where I had a scholarship from 1988 to 1994 with 50 % teaching and 50 % university courses and research. From 1995, I have had a full time position as research scientist at SINTEF[1]. The work presented in appendix D has been performed using SINTEFs tools in my spare time.

It has been inspiring to learn to know the research culture at SINTEF. This has broadened my view of science. I appreciate the inspiration and experience of my fellow researchers. I thank my former colleges from the Microelectronics Group at the Department of Informatics for encouragement. I have used the possibility of supervising several master students during this project, and I want to thank my students for valuable discussions.

I would like to thank my supervisors, Professor Yngvar Lundh and Professor Oddvar Søråsen. Whenever I have knocked at their doors, they have been positive, and a meeting has been arranged in short time. Especially the detailed help from Oddvar Søråsen with improving my scientific language has been very helpful.

I appreciate my family's acceptance of my need to finish my degree. I am also very thankful to the friends who still call me. The person who has contributed most to making this work possible, is my wife Anne. She has encouraged me in my work, and also tried to make me live a life outside with family and friends.

Joar Martin Østby

[1] The research in this position has not concerned the work presented in this thesis.

Contents

PART 1

INTRODUCTION

Chapter 1

Introduction

Developments in circuit- and device technologies continue to improve computer performance. While the basic concept attributed to John von Neumann probably underlies the majority of processors in production and use until this day, large and increasing interest and research effort aim to build even more powerful, cost-effective and compact computing systems by distribution of computing power, including specialised processing devices, into parallel processing systems. Hence many types of multiprocessing and multicomputing systems exist, and more are being designed and investigated.

SWIPP is a principle for heterogeneous multicomputing under development at the University of Oslo. "SWitched Interconnection of Parallel Processors" (which are called Compute Engines and can be widely different types in our case) is achieved with a network of switches, optical fibres and interface units called Protocol Engines. The SWIPP project relies on an expected further circuit-technology development towards a point where both powerful switches and protocol engines, respectively, can be fully integrated. The SWIPP project endeavours to contribute to a better understanding of how such integrated parts could be exploited in still more effective and useful information processing techniques.

The author has, with graduate students under his supervision, studied the implementation and behaviour of an input-buffered switch architecture. A detailed implementation has been made by layout of critical parts. VHDL code has been written for simulation at a top level and for simualtion of several switches in a small network.

The input-buffered switch can be designed with a simple switch architecture, which gives the advantage of a small area for control logic and more area left for buffer space. With this simple switch, the effort can be put in optimising for low latency at low load, and for high bandwidth. The main disadvantage of the input-buffered switch is its tendency to saturate. This behaviour depends strongly on topology, routing pattern and packet lengths.

Input buffering is well known from literature, but detailed descriptions of implementation and behaviour are not usally published. This thesis (and the master theses referred to in this thesis) gives a detailed description of how an input-buffered switch may be designed, and how performance can be improved. The thesis also gives a more thorough understanding of the limitations of this switch architecture than commonly given in standard literature, and of how their negative effects may be reduced.

The author believes that the schematics and architectural solutions given in this thesis are novel. A more detailed description of contributions and what is believed to be novel is given in section

21.5. This report focuses on the switch as an integrated component. It is the ambition of the author to show how a powerful switch would perform in a SWIPP network and how it could be built as a single integrated circuit.

Chapter 2

Background

This thesis is a part of the SWIPP (acronym for SWitched Interconnection of Parallel Processors) project of the microelectronics group, Department of Informatics, University of Oslo. One of the ideas of this project is to add increased performance to a system of computing resources by connecting them functionally closer together. The computing engines may be general computing resources (workstations, PCs, file servers) and/or specialised computing resources (vector computers, massive parallel processors). The increased system performance is attained through a distributed global resource management system and through a high performance network with effective interfacing.

Background: The steady increase in computer performance.

It is a common opinion that workstations and PCs double their performance every 18 months.[1] There has been a steady development towards smaller transistors and increased chip area, resulting in higher speed and more transistors per chip.[2] Newand more efficient architectures have been developed. This has improved PCs and workstations for general tasks as well as specialised computers dedicated to multimedia, publishing and heavy signal/graphical processing. For heavier calculation tasks a number of tightly packed processors have been put together as massive parallel processor banks and vector machines.

Reasons for connecting computing engines.

There are several reasons for putting computing engines as listed above together into systems:
• The total computing power required may be larger than any single computing engine can offer.
• The characteristic of the tasks to be performed may require a combination of specialised computing engines. Different parts of a task may benefit from being executed on computers with different characteristics.
• Resource sharing gives better utilisation of specialised resources seldom used for each single

[1] Although several system performance tests (SPECint95, SPECfp95) are commonly accepted, performance evaluation is not an exact science.

[2] The 4004 released in 1972 had 2300 transistors and a clock rate of 750kHz while the Alpha 21164 released in 1996 had 9.3 million transistors and a clock rate of 500MHz. ([23] pg. 29. fig. 1 and [95] pg. 38 Table 3.). The development of processors between has followed a steady logarithmic line with no large steps. If we consider the transistor count alone, the number has doubled every 22 months. By including clock rate and knowledge of efficient architectures we should approach a doubling in performance every 18 months.

task.
• Putting all resources together and letting tasks request resources from a "common bank" gives a better total utilisation. The variations in the demand from independent tasks will partially eliminate each other, thus giving a less variable total demand[3]. This economy of scale may reduce the total amount of computing power, memory or other resources required.
• Idle and low-loaded resources can be used to off-load more heavily loaded computing engines.
• It may be required that computing resources perform parts of tasks in different and specific physical positions in a room, laboratory, stage or building.

Examples of such systems are
• Clusters of heterogeneous computing engines put together to solve large computing tasks like weather calculations, calculation on multi-dimentional pictures, calculation on large data streams of pictures, sound or measured data which have to be served as they arrive.
• Networks of workstations and PCs for interactive use by people,
• Systems of specific and/or general computing resources used for parallel and/or serial processes,
• Laboratory measurement and processing systems,
• Systems of cameras, graphical calculation and presentation equipment and
• Combinations of these.

2.1 SWIPP — for optimal resource utilisation at a system level.

Naturally, optimal performance is a function both of the performance of each individual part and of how well each resource is utilised for the system as a whole. In the SWIPP project, optimal resource utilisation is sought in two ways:
• A global, distributed resource management system (i.e. a global operating system), and
• A high bandwidth, low latency network.

We may expand the term "computing engines" to cover other equipment capable of generating or using data in a digital format. Examples are sensors/detectors, laboratory measurement and signal/power generating equipment, cameras, monitors/screens, file servers etc. The equipment covered by the term "computing engines" have the following in common:
• They generate and/or consume data that are in or can be transferred to a digital format.
• They are physical units separated from each other, so that they can have different locations in a room, laboratory or building.

2.1.1 A global operating system.

Basically our computing engines are atomic units, i.e. each is treated as an undivided whole consisting of processor(s), memory and storage and input/output devices. It is the purpose of the operating system to schedule tasks to these computing engines as efficiently as possible. The operating system knows the general and specialised functions of each computing engine and their temporary status (load, resource queue etc.). The operating system should also as far as possible know the computing characteristics and demands of each task. Tasks are scheduled to

[3] [8] p.212: $\sigma_p = \sigma_i/\sqrt{n}$ where σ_p is the standard deviation for a population, σ_i the standard deviation for the individual elements and n the number of elements.

the computing engines depending on the requirements of the tasks and the services offered by the computing engines. *General-purpose* tasks can be scheduled for any computing engine while the *Special-purpose* tasks have to be dedicated for the specific computing engines fulfilling their need. The operating system should also, as far as possible, know in advance which resources a running task will require, so that data, code and more computing power are ready when needed. Such a global system will utilise the number of computing engines, their generality and their individuality for better performance on a system level.

Basically the operating system regards the computing engines as indivisible units. It may be advantageous to operate with other units, with groups of computing engines or parts of several computing engines together. This may be the case if they perform some often needed special functions especially well together. On the other hand, it may also be efficient to make the indivisible units smaller than a computing engine. Thus a computing engine may consist of memory pages, file records or processors which may be reserved separately. It will increase system performance if memory-demanding tasks can borrow memory pages from fast accessed neighbours instead of "swapping" memory pages to a much slower disk station. Also file storage can be done faster by splitting data into several streams stored in parallel on several disk stations.

2.1.2 A high-bandwidth and low-latency network.

Increased network performance will support better utilisation of the system resources. Increased *bandwidth* permits large amounts of data to be moved faster, thus reducing the time for which the network is occupied and shortening the time before data are available to the receiver. This will reduce the threshold above which it is advantageous to forward a special-purpose task to a computer with specific characteristics meeting those of the task. Reduced *latency* reduces the preparation time to exchange signals before larger data transfers. Shorter latency also reduces the actual transmission time of data, which is especially significant for short data transfers. Short latency also makes it possible to await data and code transfer closer to the time of use. Hence better knowledge of whether the requested data/code is really needed is attained.

Also when the smallest indivisible units are memory pages or disk station records, the system will benefit from increased network performance. Short latency and high bandwidth increase the advantage of neighbour memory swapping over disk station swapping. Increased network bandwidth also shortens the time to store/read file data. This is especially the case when the file data are divided into separate data streams to/from several disk stations.

2.1.3 Combining global operating system and high performance network.

The global operating system and high-performance network have in common that they access the computing engines etc. through the same point: a network interface card ("Protocol Engine") having fast and direct access to the host, preferably direct memory access. The interface cards have both an ASIC and a general processor with memory. The interface cards perform three kinds of functions:

- Network protocol functions,
- Distributed global operating system functions, and
- Local operating system functions.

The local operating system functions represent a possibility to offload the host computing engine as appropriate. The functions performed by the interface ASIC are those which have to be performed in hardware, those most frequent, or those where an important speed advantage over

an implementation in software can be gained. The remaining functions are performed by the general network interface processor. The flexibility of a general interface processor makes it possible to change and download protocols on need. It also simplifies the development phase when new protocols are designed and tested.

2.2 The subject of this thesis — The SWIPP network.

This thesis focuses on the network part of the SWIPP project. Clearly several network solutions may be found suitable for the SWIPP system and the SWIPP applications discussed above. In the following a short presentation of design goals believed to be attractive for such a system is made. Then some of the main characteristics of the SWIPP architecture are given.

2.2.1 Attractive design goals for a multicomputer network.

The design goals believed by the author to be most important are as follows:
a) High network performance. 4.1.1
b) Minimum interrupt of host processor. 4.2
c) Network expansion with small low-cost units should be easy. 4.4
d) Good scalability. 4.5
e) Large connectivity. 4.6
f) Large traffic flexibility. 4.7
g) Low power consumption. 4.8
h) Small physical size. 4.9
i) Reduction or avoidance of major fault situations like deadlock. 4.10

An important "design goal " which is a basis for an optimal utilisation of the network and a basis for being able to fulfilling the requirements above is:
j) Attaining as good knowledge about the expected network traffic of the applications as possible. 4.11

The design goals and how they may be fulfilled are explained in more detail in chapter 4, 5, and 6.

2.2.2 The SWIPP solution.

The SWIPP network consists of network interfaces and star switches. Chapter 3 gives an overview of the Protocol Engine (network interface) and the switch architecture. Part 3 *The Switch architecture* gives a logical description of the switch function. Packet format and error handling are also described in this part. Part 4 *High Level Simulation* contains a verification of architecture and flow control based on a description in a high level language. Part 5 contains summaries of the implementations and considerations made for the different switch modules. Part 6 contains a summary and a conclusion. The summary contains an overview of the possibility of integrating the switch on one circuit with new technologies after redesign. The appendices contain details about some of the designs and analysis which have not found their natural place in the main part.

PART 2

MICRO COMPUTING

Chapter 3

The SWIPP concept.

This chapter presents the SWIPP concept. The first part of the chapter presents an approach to how a global operating system can be used for an assemblage of computing engines. Then the functions and hardware of the Protocol Engine are presented. Finally a brief introduction to design decisions more thoroughly described in the remaining part of this thesis, is given.

3.1 The SWIPP global operating system approach.

In the SWIPP research project, the execution queue is centralised and the smallest indivisable ("atomic") resources are the computing engines.[1]

The goal of the SWIPP concept [68] is to take advantage of the cost effectiveness of a mixture of specialised and general processing elements in a network environment. The network system consists of embedded programmable microcomputers named "Protocol Engines" which are attached one to each of the processing elements named "Computing Engines"[2]. The Computing Engines may be specialised or general processors. They do not need to be programmable and they may be analogue. The Computing Engines have to fulfil only one requirement: The attached Protocol Engine must have direct access to the memory of the Computing Engine.

3.1.1 Demanding Information Processes.

Demanding processing can be found in a number of different application areas. Examples are visualisation of complex three-dimensional structures, image analysis, instrumentation and event analysis in physics research, intelligent mechanical structure management, modelling and emulation in meteorology, three-dimensional flow analysis, certain data base applications and transaction handling for telecommunication services. Storage, retrieval and processing of video, sound and other types of information earlier considered out of bounds for processing by general purpose systems, are entering the realm of such programmable processing.

[1] Appendix C describes other possible ways of using a global operating system with the same SWIPP hardware.

[2] The term "Computing Engine" may be replaced with "Host processor", "Host computer" or "Application processor".

3.1.2 General purpose and special purpose processing tasks.

A homogenous multicomputer architecture consists of a number of copies of the same processing element while a heterogeneous architecture may contain different types of processing elements. Both can achieve higher capacity than unicomputers. As technology continues to evolve, computing devices which perform specific tasks at lower cost, become available. In particular this tends to be the case for specialised tasks where special-purpose devices outperform general-purpose ones. The architecture which we are about to describe endeavours to employ specialised computing engines as part of general-purpose systems.

Most large information processes can be broken down into sub-processes or parts which we call *tasks*. Instead of placing the process on one processing element for the total computation, its tasks may be placed on different processing elements simultaneously and / or at different time. The weight put on total execution time, execution times experienced by the users and cost decides which placement is considered optimal. In SWIPP, tasks which are best undertaken by a specific processing device, are referred to as *special purpose* tasks. Tasks which can be handled by any processor are referred to as *general purpose tasks*[3]. Most processes consist of both general purpose and special purpose tasks. The engineer who needs to build a cost/effective processing system to perform a demanding process needs to put together an optimum set of processing devices for a winning multicomputer configuration. These processing devices have to be connected by a network optimised for the demanding process.

3.1.3 Distributed operating system.

Off-loading the Computing Engines from management tasks will give more freedom in the choice of general and specialised Computing Engines. Thus also non-programmable or analogue Computing Engines may be chosen.

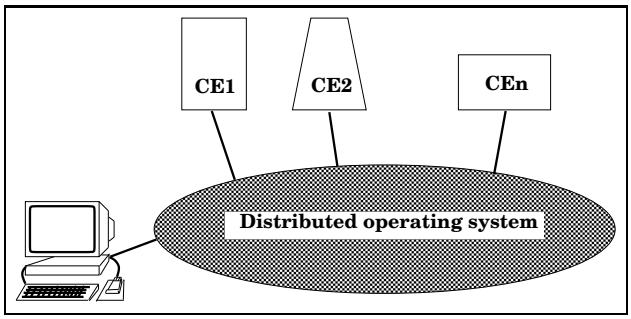


Figure 3.1: Operational control of a heterogeneous set of computing engines.

Distribution of tasks, code and data, and exchange of data between tasks, can be done by a distributed operating system. This operating system performs all functions belonging to a traditional operating system in addition to those necessary for multicomputers. The operating system is partly implemented in hardware and partly in software. To utilise the specific characteristics of the operating system, program objects matching specific Computing Engines should be identified

[3]Example: A process with matrix calculations and a general process should be placed with the matrix process on a vector processor and the general process on the general processor. Opposite placement may give similar execution time for the general process but longer execution time for the matrix process.

so that the schedule distribution of tasks can match the capability profile of the set of CEs. Such scheduling seems feasible in the case of a dedicated, stationary information process as well as in a general-purpose, less predictable environment of time sharing tasks. However, the former may be much easier to implement, hence being a more realistic first goal for system software development.

3.1.4 The SWIPP network principles.

In the following we will describe how the SWIPP network principles may be implemented.

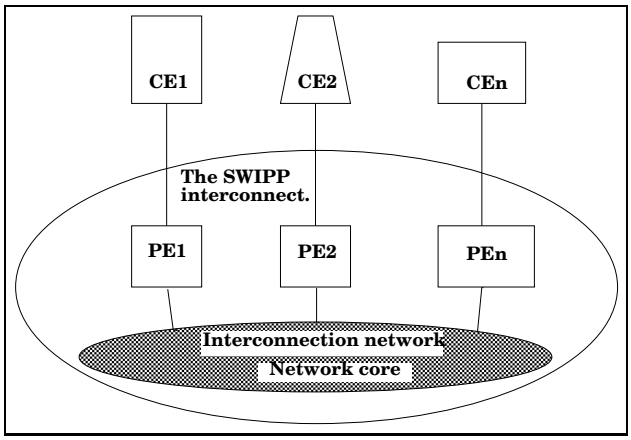


Figure 3.2: The SWIPP network principles.

Attached to each Computing Engine is one private Protocol Engine. All communication between the Computing Engine and other elements in the network passes through this Protocol Engine. All Computing Engines have an internal memory M (Fig. 3.3) to which their connected Protocol Engine has direct access. The Protocol Engine can read or write one word at a time, usually on a cycle stealing basis in a direct memory access (DMA) mode of operation. Each CE can generate a call signal to PE, while the PE can send an interrupt signal to CE.

Essentially all information transfer takes place at the initiative of the PEs. The PE controls the operation of the CE by loading programs into it and starting execution. The PE knows the CE status from reports generated by the CE. Movements of information objects between CEs, based on DMA and cycle stealing, should take place with negligible expenditure of CE computing capacity.

3.1.5 The Interconnection Network.

The Protocol Engines are transferring information objects (i.e. code or data) between each other on behalf of their respective CEs. The transmitting PE reads data directly from the CE memory. The PE itself generates the address and other elements necessary for packing of the CE data. At the receiving end the PE will unpack the data for placement directly into the CE memory.

The optimal interconnection network is decided from the traffic pattern i.e. communication pattern and frequency, degree of synchronisation between packet transmissions and requirements for bandwidth and latency. Obviously, topology will influence how tasks should be placed. Thus

tasks may be "general purpose" in one topology and "special purpose" in another.

The interconnection network will consist of switches with 16 full duplex links. One link can either connect two switches or one switch and one Protocol Engine. The links can be implemented as one pair of optical fibres or one pair of electrical cables.

An optimisation of network design implies proper choice of topology, i.e. the relative position and the connection pattern of the switches. To increase the capacity between two switches a number of channels can be connected in parallel.

More details about the network implementation are given in 3.3 of this chapter.

3.2 Protocol Engines.

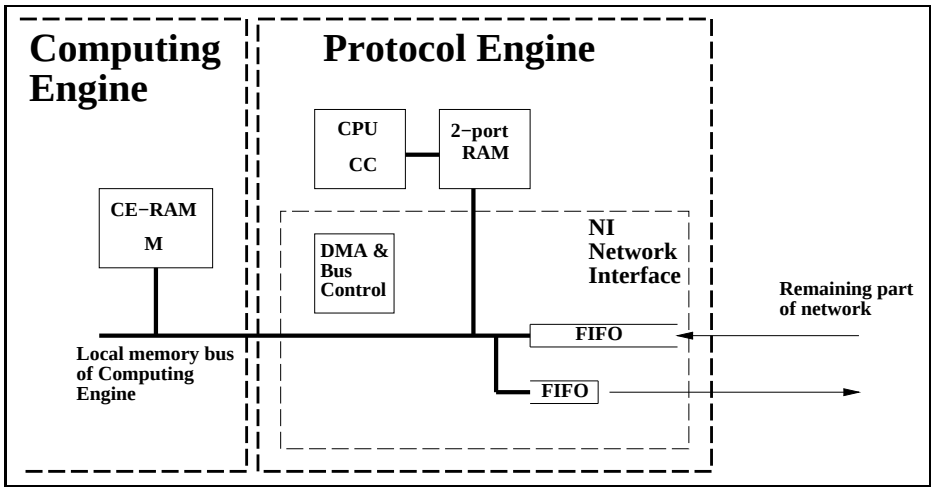


Figure 3.3: Main elements of the Protocol Engine.

Figure 3.3 shows the main elements of the Protocol Engine. The Protocol Engine consists of a general processor entitled *CC - Control Computer*, a two-port RAM and an integrated circuit: the *NI - Network Interface*. The NI contains one FIFO buffer for the incoming network link and one FIFO buffer for the outgoing network link. The NI also contains a DMA to support fast transfer of data to and from the Computing Engine memory. The two-port RAM contains communication data and communication tables.

The Protocol Engine executes two types of network communication tasks:
- Protocol Engine to Protocol Engine communication and
- Computing Engine to Computing Engine communication.

The first type of communication is network management communication. This communication takes place independently of the Computing Engines. It is used to update routing algorithms and exchange other information necessary for network management. The packets belonging to this communication are routed to and from the Protocol Engine memories.

In the second type of communication, data are transferred for the application tasks. For a globally distributed operating system as described in the previous pages, all communication is initiated by the Protocol Engines. The Protocol Engines decide where tasks are to be performed, initiate

the tasks and make sure that data and code are present. The Protocol Engine initiates other tasks when the running task has to wait for data. In such a system the Protocol Engine may be regarded as the master while the Computing Engine is the slave.

Alternative use: CE as master for more independent applications.

Probably all the hardware and some of the software may also be used in a system with more independent applications. In such a system the Protocol Engines have more limited responsibility and their software is probably faster to implement. For such a system the Computing Engine may be regarded as the master while the Protocol Engine is the slave. A data transfer is initiated by the application software with a procedure call to the Protocol Engine. The Protocol Engine will read data directly from the CE-memory and forward the data to the outgoing FIFO buffer. Received packets will be unpacked and placed directly into the Computing Engine memory. To reduce latency (section 4.1.1 of this thesis), as much as possible of packing, unpacking, segmentation, assembly, error code generation and error check, is done on-the-fly. A factor resulting in reduced performance for most systems, is the large part of the communication time during which the Computing Engine has to be involved in the communication process ("overhead": section 4.2). In SWIPP this time is reduced by leaving as much as possible of these tasks to the Protocol Engine. Thus the Computing Engine can better utilise waiting time to perform other application tasks.

Communication between the Computing Engine and the Protocol Engine takes place through memory status words. Communication from the Protocol Engine may also be initiated through interrupt signals. Thus an incoming packet may be signalled with an interrupt signal both at the start and at the end of the packet arrival. This may be advantageous for long packets.

Transmission of a packet from a Computing Engine.

A transmission of a packet from the Computing Engine has the following sequence of actions:
1: The Computing Engine signals a packet transmission request to the Protocol Engine. The signal contains directly or indirectly the necessary information about the transmission.
2: The Protocol Engine prepares the transmission by combining the received information with local information like address tables. The necessary information is transferred from the CC to the NI. When ready, the DMA reads the data out of the CE-RAM, the packet header is inserted, and the packet is delivered to the outgoing FIFO. Except when the CE-RAM bus is occupied, the CE-processor is free to execute other application tasks.
3: Depending on the parameters for the communication, the CE is notified or interrupted when the packet has been transmitted or when a receipt has been received from the receiving end.

Receiving a packet to the Computing Engine.

When a packet is received, the following sequence of actions takes place:
The Protocol Engine reads the packet header, unpacks the packet and forwards the data to the reserved memory location. Depending on the communication parameters the Computing Engine is notified or interrupted at the start and/or the end of the reception.

Chapter 6 describes some other network systems and their network interfaces.

SWIPP network	
Topology:	distributed star switches
Address format:	source routing
Switch architecture:	input buffered
Min. buffer pr. link:	960 bytes = $9.6\mu s$
Flow control:	signalled on change of status.
Packet format:	variable size
Arbitration:	fixed sequence, private to each output
Channel bandwidth:	800Mbit/sec.
Number of switch channels:	16 full duplex channels
Routing strategy:	Wormhole-like, single lane

Table 3.1: The main design decisions made for the SWIPP network.

3.3 Choices for the SWIPP network.

The main design goal for the SWIPP network is to develop a network consisting of small, inexpensive switches with as high performance as possible. Further increase of performance is attained through adding more switches and channels. Other important design goals are high connectivity and high scalability. The physical size of the network is between a room and a few buildings.

An architecture is described and partially implemented according to these design goals. The remaining part of this chapter will give an overview of the design decisions made for the SWIPP network.

3.3.1 Topology: distributed star switches.

We have chosen to use a distributed star switched topology. This choice is not unambiguous. Buses and rings have limited scalability and connectivity, but larger connectivity and scalability may be achieved through connecting several buses and rings together with bridges. The star switches require only one network element. This element in itself may be regarded as a bridge connecting several sub-networks.

For low cost, a network of switches may first be implemented with a low N_{switch}/N_{host}[4] relation (relative to the alternatives discussed later). Such a network may be regarded as a low-cost network offering a limited performance to the connected Computing Engines. In this solution the number of channels between switches is low with a high probability of contention. If the N_{switch}/N_{host} relation is increased, more channels may be used to get higher connectivity between the switches. The added channels may be used to reduce the probability of contention and increase performance. Another possibility is to dedicate channels for one or a limited number of channels to offer guaranteed bandwidth.

Thus star switches are chosen due to their ability to support networks with different sizes and different performance requirements.

[4] N_{switch} is the number of switches, N_{host} is the number of hosts.

3.3.2 Address format: Source routing.

A source-routed address format has been chosen for SWIPP. The source-routed address contains a description of the entire path in the format of a sequence of maximum 15 nibbles[5]. With this address format time is saved since table look-up in the switches is avoided. Circuitry for initiation and updating of address tables is also avoided. This address format simplifies the switch architecture, making it possible to design smaller switches and to use shorter time to establish new connections. The address format supports simpler switch architectures. However future switch architectures using rerouting or advanced buffer architectures can also be used with this address format.

3.3.3 Switch architecture: input buffering.

The switches have internal buffers.

Of the simpler buffer structures input buffering, output buffering and central buffering are most common. We have chosen to use an input buffered switch architecture. The reason for this choice is its simple architecture with a potential for design of the smallest and most inexpensive switches possible. Load saturation for some traffic patterns, the main disadvantage of this architecture, can be reduced by adding more switches and channels in parallel.[6]

3.3.4 Flow control: signalling change of status.

The buffers are protected from overflow by flow control signals between the network elements. Buffering with flow control is necessary to avoid packet loss. Typically in systems forwarding standard data traffic, execution of complex network protocols constitutes a significant portion of the communication latency. Avoiding packet loss is important to simplify these network protocols and to utilise the bandwidth. A retransmission would increase the load on the channel segments already passed.

SWIPP uses a network protocol where flow control signals are forwarded only when a transmission blockage has started or has ended. Because of the low error rate of networks like SWIPP, the flow control signals are transmitted once and do not require acknowledgements. The minimum use of bandwidth for flow control signalling allows a little higher efficient bandwidth for data transmission.

3.3.5 Minimum buffer depth: 960 Bytes.

There are mainly two factors influencing the choice of buffer size.
• The local flow control system used for SWIPP places some demands on the buffers. To secure correct function and prevent data loss, the buffers have to store a minimum number of symbols ($\approx$ bytes).
• To make temporary queuing as local as possible, the buffers should store a minimum number

[5]One nibble = 4 bit. A nibble chooses one of the 16 output channels in each switch.

[6]Basic ideas given in figure 5.8 and figure 5.9 in this thesis. By spreading switching over more levels, packets following a blocked packet will have a higher probability for being routed out of the blocked path. (Example of 64 channel shuffling network in [109], figure 9.47.)

Load	Average number of queued packets	Packet size = 64 bytes		
		No. of bytes	Time to fill/empty buffer	Max netnode distance
50%	2	128 bytes	$1.28\mu S$	100 m
60%	15	960 bytes	$9.6\mu S$	900 m
70%	100	6400 bytes	$64\mu S$	6000 m

Table 3.2: Average load in a SWIPP switch. This load is based on independent packet generation at the sources and a saturation due to the switch architecture. A packet length of 64 bytes has been selected because this is the size of a 53-byte standard ATM cell including the maximum overhead added for our network. The maximum netnode distance assumes a transport medium with high propagation speed (optical fibre) and no gate delays inside the net nodes.

of (average) packets.
Both of these factors will be discussed in greater detail later in this thesis.

The requirement for proper flow control is a function of the link bandwidth and the local flow control reaction time (flow control round time). The latter is a function of the clock frequencies in the transmitter and receiver nodes and the physical distance between the switches.

For efficient network performance it is important to make temporary queuing as local as possible. When queuing grows and influences neighbours, larger network parts may easily suffer from performance degradation. Obviously, to make temporary queuing local, the buffer capacity should be significantly larger than the average buffer space occupied. Expected queue length is a function of when new packets are released by the sources, the packet lengths, the routing pattern, the channel bandwidth and the switch architecture. Here the burstyness of data traffic has to be considered. Simulations of burstyness of local network traffic has shown buffer requirements 10 times as large as "evenly" distributed average traffic [32].

The third and fourth columns in table 3.2 show the number of bytes in a buffer and the time to fill/empty the buffer with a packet length equal to 64 bytes.

To make contention local, allow large distance between net nodes, allow many connections and large packets, buffers should be as large as possible. If the switch is implemented on one circuit, unused space should be utilised for buffer. There is no absolute minimum size, but based on table 3.2 a reasonable minimum requirement would be approximately 1 000 bytes per channel. This would allow an even average small packet load of 50 % without traffic saturation.

To guarantee that blockage will not spread, a buffer must have the capacity to store all data in the network for connections passing the buffer. To limit the data amount of each connection in the network, global flow control between source and destination hosts should be used.

3.3.6 Variable packet size

Input-buffered switch architectures may easily be designed to support variable packet lengths. Thus the Protocol Engines alone may decide whether variable packet lengths are allowed or whether a maximum of fixed packet length is practised. With longer or variable packet lengths,

		Status	Action
1	$L_{data} \ll L_{buffer}$	OK	No
2	$L_{data} < L_{buffer}$	Warning	Inspect traffic
3	$L_{data} > L_{buffer}$	Expanding contention	Global flow control
	$(L_{global-fc-data-amount} < L_{buffer})$	Increased flow control traffic	
4	$L_{pk} \gg L_{buffer}$	OK	Circuit switching

Table 3.3: The table shows a general strategy for how the network should act on different traffic situations. For little traffic, L_{data} is a packet length. When the queue lengths is larger, L_{data} is the queue length considered. In case one in the table, traffic is expected to flow, and little involvement is needed. In case two, traffic may saturate, and contention may spread. Routing algorithms should spread traffic. Global flow control may be used. In case three, the traffic level when saturation occurs, is reduced. Blockage will expand to other switches (except if the sources are one or few). Routing algorithms should spread traffic, and global flow control should limit the data for each connection. In four, when the packets are long, the channels should be reserved for one connection at a time. This will also give better utilisation of bandwidth, since less flow control is required.

bandwidth will not be spent on the multiple packet headers required when a large packet has to be segmented. Thus a little higher efficient bandwidth may be achieved. Both fair and unfair arbitration have been implemented. The author suggests unfair arbitration for new implementations. The reason for this is the need to give traffic from other switches priority over hosts. Thus traffic inside the network is given priority above arriving traffic.

3.3.7 Arbitration: Private to each output channel

Each output channel has a private arbitration block. The arbitration algorithm is simple: each input channel is given a fixed priority. The distributed arbitration supports shorter time to establish new connections. The simplicity of the arbitration makes it possible to implement a switch on a smaller area. The (possible) disadvantage of the simple arbiter is the unfairness of the arbitration algorithm. This is discussed further in chapter 13.

3.3.8 Channel bandwidth: 800Mbit/sec.

Independently of our current choice, new applications will always come, requiring more bandwidth. Thus the demand for bandwidth can never be fully satisfied.

The link between network nodes consists of one wire in each direction (full duplex). Each line contains coded data to support clock, bit, word and frame synchronisation. To reduce power consumption and simplify switch layout, it is attractive to operate on a clock rate below 800 MHz inside a switch. The internal clock rate of the SWIPP switch is 100 MHz. Thus, the single-line bandwidth of 800 Mbps between network nodes are inside the switches implemented as several parallel lines with a lower bit rate per line. We use two internal bus widths: A five-line-wide bus where data are clocked on both clock edges (200 Mbps per line), and a nine-line-wide bus with

100 Mbps per line. With present technologies ($0.5\mu m - 0.8\mu m$) 100 MHz may be considered a relatively high (but not very high) clock rate. With the technology available when we started the design of SWIPP ($1.5\mu m - 2.0\mu m$) the clock rate was at the cutting edge and design had to be done more carefully. The clock rate chosen allowed few gates between latches. Logical functions had to be incorporated into the latches. The latch in figure B.11 is an example of how this could be done with the dynamic latch in figure J.2. For the clock rates chosen, clock skew and other routing dependent time elements may be reduced to below 20% (1ns) of a clock period. For higher clock rates, routing of clock and data signals is more critical and has to be done with much more care.

High bandwidth and thus the serial (clock rate) factor of it is very important. There are several reasons why the clock rate has not been pressured to approach the limits of the technology in the present versions. To achieve higher bit rate than 200Mbps per line and a low error rate, both design experience and feed-back from measurement on produced circuits are required. Presently the project members have only limited experience and success with circuits at 100 MHz. Thus it is regarded as more important to make the total system operate, before further pressure is put on clock rate.

Stressing clock rates of circuitry and connection wires to too high values may not be a good solution for the system as a whole. This is because an increase of clock rate also increases the BER[7][8]. An increase of error rates may require more complex communication protocols adding more latency. If the error rate is high, each packet has to be collected and checked in every switch through the network ("store-and-forward"). If the error rate is lower, packet parts may be forwarded as they arrive ("worm-hole").

Our choice is to design with a bandwidth larger than the bandwidths of standard networks. The clock rate should not be higher than that we can achieve an acceptably low error rate. The error rate should be low enough to defend the choice of a worm-hole routing architecture. It should be possible to transfer ATM [82] cells as payload in our packets, and that they are transmitted with an efficient bandwidth above the 622.08 Mbit/sec ATM standard. In our network this is satisfied by a bandwidth of 800Mbit/second.

3.3.9 Number of switch channels: 16 full duplex channels

We have chosen to have a full duplex 16-channel switch as our design goal.

Why not less than 16 ?

A high number of channels gives a possibility to connect each switch directly to a larger number of other net nodes (computing and protocol engines or switches), thus reducing the number of jumps, the latency and the chances of contention. To double the bandwidth the switches may be used logically as 8×8 switches with two and two channels used together. To increase bandwidth further, other numbers of channels may be grouped together. Then the data stream has to be split at the source and merged at the receiver. A full utilisation of this requires a more complex Protocol Engine than described in this thesis.

[7] Bit Error Rate= Number of erroneous bits/Total number of bits sent.

[8] This is not the case for noise where the number of corrupted bits are linear with the number of bits in a time periode: A doubling of bandwidth will double the number of erronous bits while BER is left unchanged. The increase in BER comes from that the fast latches are disturbed by noise which is too fast to influence on slower-reacting latches. Thus the number of errors per time unit (and thus also BER) will increase with clock rate.

Why not more than 16 ?

To reduce latency we have found it important to include control logic for establishing connections and arbitration logic locally in the central switch circuit. This logic grows with the square of the number of channels. 16 channels have been found to be approximately the upper limit of what can be included with current technology.

Why 16 ?

16 is a suitable number since 16 channels are identified by 4 bits. Thus each address byte can contain the addresses for exactly two switches.

3.3.10 Routing strategy: Wormhole routing.

In SWIPP, received bytes are forwarded as soon as possible on an outgoing channel. Thus, SWIPP is using a Wormhole routing strategy with a flit size of one byte. The flow control strategy does not confirm every byte. This routing strategy supports a minimum delay inside the switch at low loads. The routing strategy trusts on a low error rate as discussed on the previous page.

3.3.11 Conclusion

This chapter has given an overview of the SWIPP architecture. We believe that this architecture would be a good choice for a network meeting our design goals (listed in the end of chapter 2). The architecture described is a good compromise between a small, simple low-cost architecture and an acceptable performance. With the present technology and design this "minimum" architecture can not be implemented on one circuit alone (Figure G.12). According to the technology forecast in table 4.2 it should be possible to include most of the switch elements (all except the opto-modules) on one circuit in few years. This is discussed in more detail in chapter 21.

Chapter 4

Performance, design goals and circuit and device limitations.

The design goals and design criteria for a multicomputer network listed (2.2.1) in chapter 2 will be discussed further in this chapter. Then some technology limitations, which influence on the capability of fulfilling the design goals, will be presented and discussed.

4.1 Network performance.

First we want to discuss how network performance may be measured and where bottlenecks have been found through measurements and analyses. Terms used in the further discussion in this thesis will be defined.

Network performance is an ambiguous parameter. A number of different figures are used to express network system performance. Bandwidth of channels, of switches or of the network as a whole are the most commonly used figures. Numbers expressing the throughput or behaviour at different traffic load levels are also much used. Others are system latency and system response time. The time the host processor (application processor) has to spend on a communication task before it can return to application tasks will be important for total system performance.

In the discussion of how performance should be measured, the competitors have especially been latency and bandwidth. The bandwidth shows the amount of data the network is capable of carrying. It has been argued that as long as the bandwidth is high enough, the latency may be hidden through clever job scheduling. Thus the most critical parameter influencing on the network performance should be the bandwidth. For real systems, latency has not been so easy to hide as expected. Hence some networks with high bandwidth, but unusable due to too long latency, have been implemented. Thus a combination of these parameters has to be inspected.

In the following, some of the performance parameters used in this thesis will be presented and defined.

4.1.1 Latency

In this thesis, the latency t_{lat} is defined as the interval from when an application initiates a network transmission until the receiving application can use the received data. It is half of the

response time, which is defined as the time between the initiation of a remote data request and the time when the requested data are available to the requesting application. Thus, it is a good indication of how the user experiences network performance. The latency is a result of network interface protocols, failure rates, bandwidths, propagation time, time to establish connections, network topology, load levels, traffic patterns, routing algorithms, etc. Hence, it should be a good indicator of the experienced performance of the network system.

To visualise different characteristics of latency, we divide it into three different components: The *propagation time* t_{prop}, the *packet-bandwidth time* $t_{pk-bw} = L_{pk}/C_{Min}$ and the *waiting time* t_{wait}.

$$t_{lat} = t_{prop} + \frac{L_{pk}}{C_{Min}} + t_{wait} \tag{4.1}$$

The propagation time is the time to forward a bit of data on an unloaded network from source application to destination application. The packet-bandwidth-time is the time to forward a packet with packet length L_{pk} through the point on the packet path with the lowest bandwidth C_{Min}. The waiting time (or queuing time) is the added time a packet has to wait for a resource occupied with other network traffic.

We also want to compare the difference in the time spent in switches and the time spent in the network interface. To visualise this difference, we divide each of the components above into a switch part and a network interface part.

Methods to hide latency.

To secure high utilisation of computing resources, processors should not stay idle waiting for network requests. The latency delay should be "hidden" by the operating system. There are two ways to do this, either or both of which may be implemented:

- Task-switching and
- Prediction of network data requests ("prefetch").

With task-switching a computer switches to another task when the task under execution has to wait for network data. Long network latency and frequent network requests would require a high number of tasks which can replace each other. This may create other bottlenecks due to switching of register and cache contents for the different tasks. For all serial tasks divided into a number of parallel subtasks there is a minimum "grain" size beneath which increased performance is not attained. The other method to hide network latency; predictions of future data and code requirements, is more difficult to implement. Prefetch messages may be put into the program code by the programmer, by the compiler, or at run time by the operating system. The amount of time required between the prefetch message and the time of use, depends on the communication latency. The code executed in this time interval may contain several conditional branches. With "lazy update", data for only one of the branches are requested. This choice may be recommended by the programmer or based on statistics gained by the run-time system from previous executions of the same code segment. This update method will result in additional data traffic to compensate for missed guesses. It will also result in increased execution time. With "eager update" advanced data reservation is performed for several (or all) of the branches in each conditional statement. Thus a considerable fraction of the requested data will never be used. This will result in a higher increase of network traffic than for lazy update. As the time between the request and the use of network data increases, the unused part of the network traffic will typically increase. Thus increased network latency will result in increased network traffic for both update methods.

4.1.2 The propagation time.

In this thesis we define the propagation time as the time to propagate a signal not only along a line or through electrical circuitry, but also through communication protocol code on a processor. The propagation time is independent of packet lengths and delays due to other network traffic. The propagation time depends on:
- network interface protocols,
- network interface clock rate,
- network topology, i.e. the number of switches to be passed,
- routing algorithms,
- wire length,
- wire characteristics,
- switch architecture,
- switch technology (CMOS, BiCMOS, GaAs etc.),
- switch clock rate.

From equation 4.1 we see that the propagation time makes up a larger fraction of the latency for small packets and/or high bandwidth than for large packets and/or low bandwidth. In practice, for large packets the picture is not completely as indicated by the equation. The reason for this is that transmission of large packets requires additional control packets to prepare buffer space etc. Thus the propagation time has to be taken into account several times. While the propagation time was less dominating in older networks due to low bandwidth, its contribution for coming high bandwidth networks has resulted in a lot of research for simpler protocols. Thus several protocols operating without copying between memory and I/O buffers have been developed. These protocols operate in a direct-memory-access mode where correction codes are generated on-the-fly and added at the end of the packets.[1]

Some example values.

Typical values for the switch part of the propagation time are in the range from $10ns$ to $1\mu s$. The lowest values are achievable in ring networks with simple arbitration strategies. For switches connecting a higher number of links, the propagation time is in the range from a few hundred nano-seconds to a few micro-seconds.

The time used in the network interface (application processor time included) will in most cases be the larger part of the propagation time. General communication protocols like IP (Internet Protocol [46]) may require up to $1000\mu s$[2](IPv4). New, more efficient versions of these protocols which use $250\mu s$ have been developed[3] Myrinet [10] has developed an API (Application Programming Interface). It is difficult to figure out the latency from the papers describing the protocol, but it looks like it is less than $90\mu s$. Specialised protocols designed for specific computers may achieve lower values. Thus Digital Research [88] has developed protocols requiring only $65\mu s$ for communication between their workstations. The NOW team at Berkeley ([3]) has developed a protocol set named *Active Messages* [69] with a protocol latency equal to $15\mu s$. They claim that

[1] Further discussion with references in the next subsection and in subsection "Old protocols contribute.." under 6.10.2 of this thesis.

[2] According to [102] page 3, an IP packet has to pass the system bus 6 times in the typical old implementations of IP.

[3] Analysis of IP optimisation in [101] [102] and [100]. Examples of new protocols are IPv6/IPng [50] and RTP [51].

they are going to reduce the latency to $10\mu s$. The line of high specialisation/low propagation time may be drawn all the way to "dumb"-circuitry like cameras requiring very little time for communication protocols.

4.1.3 The packet-bandwidth time

The packet-bandwidth time is equal to the packet length in information units (bits or bytes) divided by the execution speed of the resource (typically a channel) in information units per time unit (bits/sec or bytes/sec).

A packet length may be chosen based on:
- application characteristics,
- efficient lengths for buffer design, and
- efficient length depending on network traffic.

The channel bandwidth is a product of the parallel and serial bandwidth:
- The serial bandwidth (clock rate) is a function of technology and architecture.
- The parallel bandwidth per channel is a function of the total pin count (number of circuits and packet technology) and how many channels the pins have to be shared between.

Bandwidth determined by application memory speed.

In most previous and present systems, network communication has taken place through dedicated I/O-circuits. This has been a suitable solution for computers of yesterday connected by low bandwidth networks (i.e. 10Mbps and below). These I/O-circuits could be bottlenecks which would significantly reduce performance in coming computer systems. Direct-Memory-Access methods are expected to become the standard for such high-end computers. A network matching Direct-Memory-Access systems should have a bandwidth similar to the bandwidth at which data are written to or read out of the memory. As an example, a 64-bit-wide data bus operating at a speed of 100MHz would have a data rate of 6.4 Gbps. This is a high number, for which a single-wire solution requires expensive components, while a number of less expensive components are needed for parallel wiring. Since most applications would use time between network accesses to manipulate on received data and data for transmission, the network connections would be used only a part of the time. Thus, it may be more cost-efficient to use a less expensive somewhat lower bandwidth. Whether the bandwidth should be pushed the last bit at a high cost, will depend on the application and how the users evaluate the importance of this last additional increase of performance.

Some example values

Optimal packet sizes are chosen based on applications and network. Examples used in the following are 53byte (the ATM packet size [82]), 200byte[4], 8kbyte (medium file packet) and 1Mbyte (large file).

[4][33]: assumed "typical upper limit in cluster traffic", [3]: "95 % of all packets in a typical local network" has a length less than 200 bytes

For the type of networks we are discussing, typical bandwidth values are 155Mbps (ATM [82]), 622Mbps (ATM [82]) and 8000Mbps (SCI [48]). These values may be compared with the slow 10Mbps standard Ethernet.

4.1.4 The waiting time.

Since this thesis is mainly focusing on switch architecture, we may divide the waiting time (or queuing time) into the two following parts:
- a part mainly *independent* of switch architecture and
- a part strictly *dependent* of switch architecture.

The first part is depending on the time between packet arrivals, the variation in this time like tendency for bursts, the destination of the packets, the packet lengths and the channel bandwidth. This part of the waiting time can only be eliminated completely through strict synchronism in the application code. Since this seldom is possible or advantageous, some waiting time of this kind must always be expected. It may be reduced by distributing traffic load as evenly as possible on the available routing paths. This is depending both on the network topology and the efficiency of the routing algorithms. The routing algorithms, executed by the Protocol Engines, may be improved by collecting measured delays from the different paths. Another way to reduce this waiting time is through increased channel bandwidth.

The second part of the waiting time comes from local blocking inside the switches. This additional delays come when packets due to the switch implementation can not be forwarded to idle outgoing channels. Waiting time due to switch architecture may, beside changing the architecture, be reduced through increased bandwidth.

The relation between the packets arriving at a buffer or channel is initially decided by the application and may later be changed by the network. If all computing nodes execute the same application, the execution at the different computing nodes may be synchronised to initiate network traffic with a fixed phase relation. If the computing nodes perform several application tasks, the requests may be independent. The network influence may "queue" several initially simultaneous requests nicely into a sequence when they arrive at the resource. In this case the expected waiting time based on the packet transmission from the sources, are distributed over several switches.

Some example values.

We simplify and presume an ideal switch, i.e. we are using the part of the waiting time independent of switch architecture. If packets for a channel arrive completely independently and packet lengths vary with a Poisson distribution, the average waiting time will be at least $t_{wait} = (1/3)t_{pk-bw}$ with a load of 0.25, $t_{wait} = t_{pk-bw}$ with a load of 0.5 and minimum $t_{wait} = 3t_{pk-bw}$ with a load of 0.75.[5]

[5]These calculations are based on equation 1.59 from [61] where $W = (\rho/\mu)/(1-\rho)$ for a M/M/1 system. Here ρ is the load, while $1/\mu = t_{wait}$.

t_{pk-bw}					
Bandwidth:		10Mbps	155Mbps	622Mbps	8000Mbps
Packet	53 Byte	$43.2\mu s$	$2.8\mu s$	$695ns$	$53ns$
	200 Byte	$160\mu s$	$10\mu s$	$2.6\mu s$	$200ns$
length:	8 kByte	$6.4ms$	$412\mu s$	$103\mu s$	$8\mu s$
	1 MByte	$0.8s$	$52ms$	$13ms$	$1ms$

Table 4.1: t_{pk-bw} for some packet length and bandwidth examples. The line separates the values below and above $250\mu s$.

4.1.5 The relation between the latency example values.

By comparing the example values of the latency components given in the previous pages, we find that their contribution varies considerably. Knowledge of their relative contribution is important in the design process, so that the designer can place most effort where there is most to gain.

First we need to know the typical values for the t_{pk-bw}.

We see from table 4.1 that a network propagation time of $200ns$ is shorter than most t_{pk-bw} values. Only the t_{pk-bw} for the smallest packets at the highest bandwidth is comparable. With a network interface propagation time from $250\mu s$ and upwards for general computers, delays through the host and network interface will be the main contributors to the latency (except for the largest packets and the lowest bandwidths, for which the network bandwidth will contribute considerably).

Based on the example values above, we can draw the following conclusions to where effort should be put to reduce latency: If we expect large packets, the bandwidth should be increased. This will reduce both t_{pk-bw} and t_{wait}. If we expect smaller packets, the network interface propagation time must have higher priority. In any case, for the numbers given, the network part of the propagation time gives the smallest contribution to the total time, and should be given the lowest priority when effort is put into improving network performance.

A conclusion from the last relation between bandwidth and latency is that high clock rate should be stressed to move large amounts of data rather than to move data fast. This influences switch and network architecture. An example is to increase the number of latches to be passed between "A" and "B" (pipelining). The disadvantage of this action will be an increase of propagation time between "A" and "B" while the advantage will be that more data are simultaneously on the way.

4.2 Host computer occupancy time or "host overhead time".

Also important for the system performance is to what extent the host computer can be released from the communication tasks. The host computer is influenced either by participating in the execution of the communication tasks or by forwarding data to/from the network. The main ways to reduce the involvement of the host computer are:
- Using simpler communication protocols, and/or
- Moving communication tasks from the host computer to the network interface.

- Accessing the host memory with the maximum memory access rate (DMA), and/or
- Transfering data between network and host memory on a cycle-stealing basis if sufficient.

For some new network systems with large network bandwidth, the overhead has become a significant part of the latency. High overhead reduces the time for which the host computer can deal with alternative application tasks during a communication delay. Examples of such utilisation of the computing engine was given in *Methods to hide network latency* on page 21. A typical communication overhead of $250\mu s$ corresponds to 25000 cycles on a 100 MHz computer. In [33] an example is given where a molecular-modelling program is running on two computing clusters connected by a standard network. Due to high overhead they use 75% on communication and 25% on useful work. Thus more than twice as much hardware does half of the work of a single cluster. In [33] it is stated (but not documented) that historical data show that performance of operating system services improve less than application performance as microprocessors speed up. The author of this thesis shares this impression. Reasons for this may be that operating systems have to be compatible with older versions and thus more conservative. Applications are more independent and can therefore be redesigned from the beginning.

Thus, to increase performance, it is not only important to reduce network latency but also to reduce the involvement of the host processor.

4.3 Efficient bandwidth and utilisation gaps.

When a resource like a bus or channel is going to be used, some time is required for preparation. Preferably this preparation should take place while the previous user is still using the resource. If some of the preparation has to take place while the resource is not in use, the efficient bandwidth will be reduced. The non-overlapping preparation time between resource utilisation is entitled $gap : t_{gap}$.

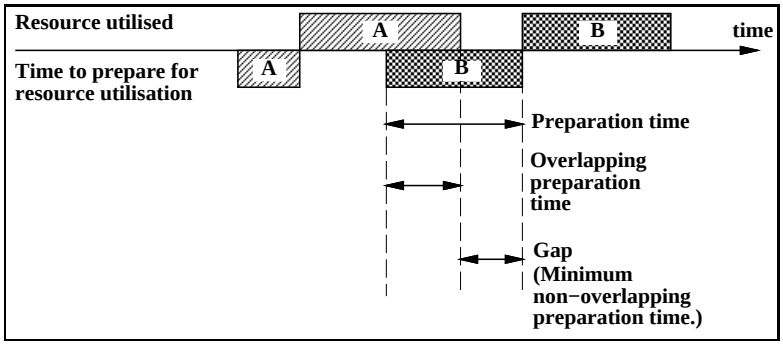


Figure 4.1: Reduced efficient bandwidth due to non-overlapping preparation time: gap.

Figure 4.1 illustrates the non-overlapping preparation time. If the resource is a bus or channel with a bandwidth C, an average packet length L_{pk} and a non-overlapping preparation time t_{gap}, the efficient bandwidth C_{eff} may be expressed by the following equation:

$$C_{\text{eff}} = \frac{L_{pk}}{t_{gap} + \frac{L_{pk}}{C}} = \frac{L_{pk}/t_{gap} \cdot C}{C + L_{pk}/t_{gap}} \tag{4.2}$$

The relation between C and L_{pk}/t_{gap} is important. If the packet length L_{pk} is kept long enough, or the t_{gap} is kept short enough, $C \ll L_{pk}/t_{gap}$ and $C_{\text{eff}} = C$. If, on the other hand, C is increased (or L_{pk} reduced or t_{gap} increased) so that $C \gg L_{pk}/t_{gap}$ the efficient bandwidth will approach L_{pk}/t_{gap} independently of further increase of C.

The increased bandwidth and the use of short packet lengths in dense networks have increased the dependency of the efficient bandwidth on the non-overlapping preparation time. Inside switches, buffering and architecture design can make the non-overlapping preparation time for an outgoing transmission acceptably short or even zero. The problem is larger at the network interfaces, where the non-overlapping preparation time may be very significant. With a non-overlapping preparation time of $250\mu s$, an average packet length of 200 bytes and a basic bandwidth of 1Gbyte/s, the efficient bandwidth is $6.3Mbps$. Specially designed network protocols for homogenous computers may achieve a shorter preparation time. Researchers at Digital claim ([88]) that they can push the overhead down to $65\mu s$ for their machines. This indicates an increase of efficient bandwidth to $24.5Mbps$ which is still far below the maximum bandwidth of ATM [82] and SCI [48] networks.

The concept "efficient bandwidth" is used in chapter 6.

4.4 Easy expansion with small low-cost units.

Preferably a network should be built from small, inexpensive elements. When this is the case, changing or increasing demands can be fulfilled in small steps. All parts should be able to contribute to improvement of network performance after an expansion of the network or change of topology. Change of network topology should be discovered by the protocol processes themselves. It would also be an advantage if network parts could be added, removed or replaced with limited disturbance of the network parts not involved.

4.5 Good scalability.

This design goal is partially over-lapping with the previous. In addition there should be very high or preferably no upper limit to the number of hosts and switches for the software and hardware.

4.6 Large connectivity.

It should be possible to connect the network in a large range of topologies, both regular and irregular, and to mimic known efficient topologies. A minimum connection solution, offering a fair performance at a low cost, should be possible. Network elements can be added to increase bandwidth, reduce latency or reduce blockage conditions. Direct channels can be connected, reserved for dedicated connections to guarantee bandwidth or a maximum limit to latency.

4.7 Large traffic flexibility.

The network hardware should make it possible for protocol software to support a large range of different traffic: Long file transfer, circuit switching for continuous data transmissions, short urgent packets and data requiring guaranteed bandwidth or a maximum limit to latency.

4.8 Low power consumption.

Since high power dissipation is a technology limitation, it is discussed in more detail in 4.12.2. As a design goal, low power consumption is attractive for saving of energy. Low power consumption also results in lower temperature. This increases the lifetime of the switches and makes it easier to place them between people.

4.9 Small physical size.

Switches with small physical size are advantageous for several reasons. They normally require less power. They are less expensive in large volumes. They are easier to handle and can more easily be placed out.

Small physical size is achieved through integrated circuit design and use of multichip modules. For connection between net nodes, optical fibre is the transport medium requiring least space.

4.10 Reduction or avoidance of major fault situations like dead-lock.

The network topology, switch architecture and communication protocols have to be designed with minimal probability of the most performance-degrading fault situations. The fault with the potential of the total paralysis of network functions is deadlock, i.e. a blocking, circular dependency of resources. Topologies with low probability for deadlocks may be implemented logically through the routing algorithm[6] or physically through the network topology. Time-out[7] may be used for the few dead-locks still occurring, and should only be used in this situation.

4.11 Knowledge about application traffic characteristics.

For initial design of an optimal network as much knowledge about the application traffic as possible should be gained. The knowledge has to be constantly updated by taking statistics of the network traffic. Traffic information required is:
• Identity of communication partners,
• How often they communicate,
• Dependence (or degree of synchronism) between packet transmissions at different communication pairs, and
• Message lengths and their variation.
Another interesting factor is
• Whether and how new traffic may change due to the network performance.
In parallel systems where fast network response reduces the size of subtasks found efficient for

[6] If altering of bits due to electrical noise did not occur, routing algorithms could have prevented dead-lock completely. Since, in real networks, electrical noise occurs, dead-lock may occur if the hardware and topology allows it (like in networks with full duplex connections, where the connections are small circles).

[7] Time-out occurs when the foremost packet in an input channel has not propagated for a specified amount of time. At time-out this packet will be dismissed.

Year	Smallest feature, μm	Dynamic RAM		Microprocessors			Wiring levels per chip	I/O per chip
		Chip size, mm^2	Billions of bits per chip	Chip size, mm^2	Millions transistors per cm^2	On-chip clock MHz		
1995	0.35	190	0.064	250	4	300	4-5	900
1998	0.25	280	0.256	300	7	450	5	1350
2001	0.18	420	1	360	13	600	5-6	2000
2004	0.13	640	4	430	25	800	6	2600
2007	0.10	960	16	520	50	1000	6-7	3600
2010	0.07	1400	64	620	90	1100	7-8	4800

Table 4.2: Roadmap projections for semiconductor technology. Redrawn from [31] p. 53.

distribution to other computing elements, the increased performance may result in increased network traffic.

The characteristics of the network traffic will influence
• how computers and network switches should be connected
• where increased bandwidth is required and
• where direct connections should be used to reduce latency and contention. The network traffic characteristics will also influence the choice of packet lengths and routing pattern. If the routing algorithms are adaptive, measured traffic characteristics will result in new routing patterns. These new routing patterns will influence back on the new load pattern.

4.12 Main circuit and device limitations.

Technology puts limitations to how a multicomputer can be implemented. Some parameters which have to be considered are:
• Complexity in number of gates per chip,
• Power density and dissipation,
• Choice of technology for a switch architecture,
• Architectural decisions giving optimal performance for the circuit area,
• Pin number limitations, and
• Fibre speed and transducer types.

Present and future technology limitations.

The development of semiconductor technology, knowledge of efficient switch and network architectures, semiconductor design and simulation software during the past few years has been impressing. As shown in the technology forecast by the US Semiconductor Industry Association in table 4.2 (from [31]), new technologies with higher number of transistors per area, larger circuit sizes, more pins and faster clock rates are expected to be developed in the next fifteen years. The table shows an expected 2.5 times increase of chip size and an increase in number of transistors with a factor 22.5 during the next 15 years. This gives 55 times as many transistors on a circuit. Although, the technology in table 4.2 is the leading edge technology not available for the average

designer, a similar improvement may be expected. Today the best technologies available to an average designer through commercial process houses are $0.5\mu m$ and $0.8\mu m$ technologies and chip areas of $100 - 150mm^2$. For our switch and network purposes new technology achievements may be utilised in different ways. Increased clock speed and pin counts may be used to increase the total switch bandwidth. This may give more physical channels, more bandwidth per channel or both. Increased pin count may also be used to keep bandwidth up, while reducing the power consumption. For switches designed today, many designers do not consider pin count to be an important limiting factor. Several serial technologies exist offering bandwidths around 1Gb/s for CMOS and BiCMOS ([12]).

The technology parameter discussion above was for CMOS technology. CMOS is the technology offering the highest density of transistors, the lowest power consumption and the lowest price. It is the dominating technology with regard to the number of designs and the total volume of circuits processed. It can be used for analogue and digital designs. Steady development of finer transistor CMOS structures allows increased clock speed making CMOS conquer application areas ruled by other technologies. The main competitor of the CMOS technology is another silicon technology, BiCMOS, containing both CMOS and bipolar transistors. Bipolar transistors offer a higher clock rate and a larger driving capability than CMOS transistors. A third competitor is the Gallium-Arsenide composite, which has found its major use for the fastest transistors, in optical systems and micro-wave systems.

4.12.1 Complexity in number of gates per chip.

The number of transistors per circuit decides what architecture can be implemented and thus the performance of the circuit. This number has been steadily increasing through the years, both due to reduced transistor size and because of better yield for larger circuits.

At the system level, larger circuits and increased density of each circuit may be used to:
- reduce the number of circuits and the total physical size of the system,
- implement more advanced architectures offering better performance,
- reduce the power consumption (explained in 4.12.2),
- increase the clock frequency (shorter delays between circuits),
- reduce the cost, and
- combinations of these.

At the host network interface, increased transistor density results in that more complicated and better protocol engines may be included on smaller area. In network switches, the largest advantage is that larger buffers may be included. In systems without flow control, increase of buffer size will increase the amount of traffic before packets are dismissed. In systems like SWIPP with net node to net node flow control, larger buffers increase the traffic level before buffer contention influences neighbour net nodes. When large buffers are allowed, buffer space may be reserved for groups of connections or for each individual connection (host to host virtual channel) resulting in an isolation of contention to less host to host connections. In this case, further increase of the number of buffer segments allows a larger number of connections (virtual channels).

4.12.2 Power density and dissipation.

High power consumption results in high temperature which reduces performance and increases the failure effects in electrical circuitry.Temperature may be reduced by better conduction of the heat through heat sinks and heat conducting substrate, and/or by changes of circuit architecture resulting in less power consumption. The dynamic power consumption for digital circuitry may be expressed as ([6] pp. 440-441):

$$P_{dyn} = \frac{1}{2} \cdot N_D \cdot Vdd^2 \cdot C_L \cdot f_{clk} \tag{4.3}$$

In eq. 4.3 f_{clk} is the clock rate, C_L is the average capacitive load per transistor and Vdd is the power voltage. N_D is the number of transistors switching on a clock edge. In 20.1, the same equation is used. In eq. 20.1, figures for the elements of the equation have been found, based on simulations on several designs. From equation 4.3 we see that the power consumption can be reduced through a reduction of supply voltage. This has the cost of lowering the noise margins. This has the energy- economic effect of making it possible to increase the usage time of equipment powered by batteries. The steady development of reduced transistor sizes allows transistors to operate at higher speed and increases the number of transistors per area. This may increase the power consumption per area. This increase may be avoided through system design, hardware architecture, choice of logic family or, as stated above, reduction of the supply voltage. The last alternative gives a square effect, since the power consumption is a function of the square of the supply voltage.Through these measures, the power consumption can even be reduced.

Power reduction in switches.

This thesis focuses on the switch architecture. The switches have circuitry for connecting each input channel to each output channel. Most of the time, an input channel will not be connected or will be connected to only a smaller number (usually one) of the output channels, hence most of the logic will be inactive. Thus, logic families with small power consumption in inactive states may be preferred.

The other power saving technique, reduced power voltage, has some interesting prospects in communication systems, where bandwidth is typically more important than clock rate. Thus a reduced clock rate may be compensated by increased bus width. For fixed bandwidth, the product of the parallel capacity (N_P = bus width) and serial capacity (C_s = bit/sec = $1/t_{clk-period}$) per line of a channel is constant. The clock period, $t_{clk-period}$, should be expected to have some linear relation to the signal raise (or fall) time (t_r). Thus the parallel capacity, N_P should increase linearly with t_r. If the power consumption of each serial line is represented by P_S we find the total power consumption to be (see also table 4.3):

$$P_{tot} = N_P \cdot P_S = \frac{C_1}{(Vdd - Vt)} \cdot (C_2 \cdot Vdd^2) = \frac{Vdd^2}{(Vdd - Vt)} \cdot C_1 \cdot C_2 \tag{4.4}$$

Thus the power consumption will shrink almost linearly with the voltage supply, but the switch can still be capable of offering the same bandwidth.

Typically, small, distributed switches would have to dissipate less power than the larger ones. There are two reasons for this: To reduce additional size and noise, they should operate without a fan. They may come in closer physical contact with the users and should not be too hot.

Supply Voltage	Delay $T_d(= N_P)$ $\frac{C_1}{V_{DD}-V_T}$	Line power P_S $C_2 V_{DD}^2$	Bandwidth power $P_{tot} = N_P \cdot P_S$ $\frac{C_3 V_{DD}^2}{V_{DD}-V_T}$
5	1	1	1
3.3	1.6	0.44	0.70
2.5	2.3	0.25	0.56
1.5	4.5	0.09	0.41
1.0	9	0.04	0.36

Table 4.3: The same bandwidth may be implemented with different products of serial and parallel capacity. A reduction of the power voltage reduces the serial capacity. The table illustrates that lower power voltage results in a reduction of the total power consumption while the same bandwidth is maintained. The second column shows the delay expressed as raise/fall time, and the third column the power consumption per line. The last column shows the power consumption when multiplied up to the reference bandwidth. Delays and power consumption are given relative to the values for $V_{DD} = 5$ Volt. The threshold value V_T has been sat equal to 0.5 Volt. In general the minimum power supply giving the minimum power consumption for the reference bandwidth is equal to $V_{DD} = 2V_T$. C_1, C_2 and C_3 are different constants. A reduction of the serial capacity also has advantages like simplifying routing of clock lines.

4.12.3 Choice of technology for a switch architecture.

At a technological level we have a trade-off between high clock rate technologies like GaAs and ECL and the CMOS technology with a lower clock rate. The GaAs and ECL transistors require more space per transistor and per memory element than CMOS.

From the discussion on page 25 it was clear that bandwidth was the most important parameter to put pressure on to achieve better performance. Propagation time (i.e. clock rate) was less important. The argument about bandwidth made in 4.12.2 is also relevant here: The bandwidth may be kept high by increasing the bus width when the clock rate is reduced. Thus, regarding bandwidth, CMOS may compensate for the higher clock rate of the other technologies. When it comes to propagation time of single data and control bits CMOS will be slower. The important functions giving the differences between the technologies are buffer implementation and interfaces to single high speed serial lines to/from coaxial cable or fibre. Although the speed of CMOS is increasing, CMOS will have a lower clock rate and current driving capability than the other technologies. On the other hand, CMOS has a much higher memory density than the other technologies. The choice of technology does not influence on the required buffer space. This is because the amount of data stored depends on network traffic (decided by applications and routing) and bandwidth (which can be supported independently of technology) and not on clock rate. Since large buffers are important for system performance, buffers should be implemented with pressure on high density. We find that no technology is number one in all areas.

4.12.4 Architecture decisions giving optimal performance for the circuit area.

The most difficult task is to find the switch architecture which gives the optimal performance for the available chip area.

Design elements to be considered are:
- The number of segments the buffer is divided into,
- The reservation strategy for the buffer segments,
- Address tables,
- Type of switch matrix(es),
- Packet format,
- Address format,
- Flow control strategy,
- Arbitration strategy,
- Bandwidth and
- Coding format for transmission between net nodes.

All the design elements listed will influence the utilisation of the available circuit area. They influence the effective bandwidth, propagation time, waiting time and thus the performance. These design factors will be discussed in more detail in the following chapters.

4.12.5 Pin number limitations.

New expensive technologies offer 1000 pins per circuit and above. According to the US Semiconductor Industry Association (table 4.2) the number will continue to increase to five times as many pins in 15 years. Limitations due to packaging of circuits may be avoided by mounting naked circuits. With flip-chip mounting, i.e. active surface facing the substrate, connection bumps may be placed everywhere on the circuit. Circuits with high pin numbers require more expensive packaging, more expensive mounting and a more expensive substrate. More expensive mounting results from that smaller sizes require more precise equipment and that the failure rate for each connection has to be reduced to keep the same success rate for the total circuit. For SWIPP the number of pins required is smaller than the maximum number possible. Some of the pins have to be used for power connection. If the circuit generates much electrical noise, the number of power pins has to be increased. Some pins will be used for clock signals. Most of the remaining pins should be available for data signals. If we denote the total number of data pins N_D and have an equal number of input and output pins, the maximum switch bandwidth is $(N_D/2) \cdot f_{CLK}$ where f_{CLK} is the clock frequency. The N_D pins may be used for many narrow channels or a few wide channels according to the following equation:

$$N_D = N_C \cdot 2 \cdot N_W \tag{4.5}$$

Here N_C is the number of channels, 2 comes from one input bus and one output bus for each channel, and N_W is the width of each bus. Increased bandwidth from wide buses (N_W) results in shorter time for a packet to pass a point. Increased number of channels (N_C) may be used to increase the connectivity between distant nodes, hence reducing the transmission time through reduced latency. Both alternatives for usage of increased pin-count will reduce the load on the network. Thus the network queuing time will also be reduced.

The number of pins chosen for the SWIPP switch is a tradeoff between cost and desired bandwidth. The cross-over point between cost and switch performance is a decision which has to be taken by the designers.

This subject is discussed further on and from page 348.

4.12.6 Fibre speed and transducer types.

There are several possibilities for the implementation of the links between net nodes. The most common choices are buses (when on the same circuit board), cable of parallel twisted pairs (for a few meters), serial electrical coaxial cable and optical fibre.

Since the coaxial cable and the optical fibre transfer all data on one "line", the bit rate has to be high. To support high baud rate and few errors, data should be transmitted using a coding scheme. Additional circuitry will be required at the transmitting and receiving ends. Preferably this circuitry should be integrated together with the remaining part of the switch.

Compared to the alternatives, optical fibre may be regarded as an attractive solution, due to the smallphysical size, the high bandwidth, the fast signal propagation and the low loss in the cable. The disadvantages are higher cost and high energy loss in the source connection point of the fibre. At shorter distances, less expensive, lower quality fibers may be used.

A solution may be to terminate the optical fibre directly on the silicon circuit. This will reduce the number of components required and the number of pins, and may reduce cost[8].. To use the silicon circuit as a photodetector, the ligth should have a wave length close to $800nm$. In this area, silicon has a quantium efficiency close to 100% ([106] p. 282). For several reasons the light source has to be a separate device. To make a light source, other processing steps have to be taken than those used for a standard digital or analog process. Due to indirect band gaps, silicon has a high loss and is a bad light source. It may be preferable to have the possibility to choose between less expensive LEDs for short distances and laser diodes for longer distances.

A flexible solution could be to integrate the encoders and decoders on the switch circut and let the choice of transport medium be decided by external devices. Thus through the choice of external devices the following transport mediums could be used:
- optical fibre and lasers for very long distances,
- optical fibre and LEDs for long distances,
- coaxial cable for medium distances,
- twisted pair for short distances, and
- direct bus when on same circuit board.

The logic of the SWIPP encoders and decoders are presented in chapter *11, The Optical Module*, while the implementation is presented in chapter *19, Implementation of the Optical Module*. Chapter 11 also contains a deeper discussion of link medium than presented here.

[8]Typically a smaller number of components and a smaller number of pins give lower cost. But, on the other hand, mounting and substrate processing may be more expensive. The total bill for large quantities is difficult to estimate. This requires more time and resources than this author has available. It is the authors guess that the cost will be smaller.

Chapter 5

Network systems, topologies and switch architectures.

To understand some of the characteristics of the SWIPP network, principles for networks with three different sizes are discussed in this chapter. According to the classification given, SWIPP is a Local Area Network. Three system applications of Local Area Networks are presented. A discussion of network topologies, simple switch architectures, buffer management, flow control and packet strategies follows. The efficiency of fixed packet size is also discussed.

5.1 Network classification based on physical size.

Networks for data transfer range from dense networks connecting a high number of processors within a small physical space to large networks connecting central data switches in different continents. Obviously these networks have different characteristics. We will in the following divide this range of networks into three categories which are compared to each other. According to the classifications in the following, SWIPP is primarily a Local Area Network (LAN), but may also be considered attractive for some Massive Parallel Processors (MPPs) systems

5.1.1 Long distance (tele)communication networks.

These networks, commonly entitled Wide Area Networks, interconnect central data exchanges from several kilometres to continents apart. The long distances make the physical links expensive resources which have to be utilised as effectively as possible. These physical links typically have large bandwidth (bits transferred per second) and long propagation delays. The transmission rate is often stressed, and the acceptable error rate higher than what can be achieved by smaller networks..[1] In networks with high error rates, each switch receives the entire packet and checks it for errors before the packet is forwarded to the next switch (store-and-forward).

Each physical link often contains a number of time multiplexed logical channels. Because the links are expensive, complex switch architectures are chosen. Thus the utilisation of the link capacity is pushed as high as possible. The main switch characteristics influencing this are the

[1] For long-distance communication the acceptable error rate is 10^{-10} ([82] pg. 36. For local area networks error rates as low as 10^{-15} can be achieved ([90]).

internal switch matrix and buffer managing politics. To simplify the switch architecture, fixed packet sizes are preferred.

A hot topic for this kind of networks is an ATM (Asynchronous Transfer Mode) protocol defined by the International Telecommunication Union, ITU, (commonly and in the following referred to as *the* ATM protocol). The ATM [82] protocol defines transmission rates, data lengths and data formats. The most popular bit rates defined in the ITU ATM are 155.520 Mbit/sec and 622.080 Mbit/sec. The ITU ATM protocol is designed as a compromise between the telecommunication and computer industries to support the needs of both industries.

5.1.2 Massive Parallel Processor (MPP) Networks.

In MPP systems, different parts of the same programs are executed in parallel on different processors. Short communication time is a very important part of the system performance. Typically, this is achieved through simple network protocols and fast network hardware. To simplify network protocols, the MPPs contain homogeneous processors, the protocols are optimised for (a limited set of) applications and require a network with very low error rate. Low network latency and high bandwidth are very important elements of the total system performance. Thus processors are placed physically as close to one another as possible. This makes it easier both to reduce transmission time and to increase bandwidth. The processors are often placed in a regular structure like an n-dimensional matrix (hypercube) or a k-ary n-cube. In many cases the processors are integrated together with a switch unit. The switches often have a simple architecture with buffers at the input ports (Mosaic [17], Transputer [14]). Switches and channels are considered inexpensive so load saturation is avoided by adding more switches and channels in parallel. High performing parallel processing systems are usually designed by one vendor with a number of processing elements of the same type running a specialised operating system. Such systems can not easily benefit from recently developed processors and typically use processors a few years old. The main reason for this is probably that it takes more time to develop parallel operating systems than single processor systems for new processors.

5.1.3 Local Area Networks.

Local Area Networks (LANs) operate on intermediate distances. LANs are used to connect "Computing Engines" within a building or storey. Computing Engines may be workstations, PCs, printers, laboratory instruments, disk stations, huge parallel engines, scalar processors, monitors etc. Such a network typically has to be dynamic, allowing equipment to be disconnected, moved around and connected again.

Until recently buses or cables have, due to their simple topology, been dominating as local area networks. Bus and cable networks have some disadvantages. One is access control which limits the effective bandwidth and the length of the bus/cable and results in increased latency. An Ethernet cable with a maximum bit rate of 10 Mbits/s can have a maximum length of 250 meters. Demands for higher bandwidth have resulted in ring topologies like FDDI with a bit rate of 100 Mbits/s. Ring topologies have a number of advantages over buses: Only one transmitter on each ring segment simplifies the access control. This gives a potential for higher bandwidth, longer cable segments and shorter latency. Both bus and ring topologies suffer from that the transport medium is shared between all connected nodes. This gives a high number of communication pairs competing for each link segment and a lower share of the bandwidth for each pair. Thus, to compensate for the lower share of the bandwidth available for each connected node, the total

bandwidth has to be high.

The Client-Server concept.

A popular concept used for LANs connecting workstations and/or PCs is the Client - Server concept. The network consists of a number of workstations/PCs, in the following titled "clients", connected to one or a few storage stations, titled "servers". The servers have a large disk capacity and a large memory. The disk stores common and private data accessed by the different clients. The server memory operates as a cache for the most used disk pages. The clients may be disk-less, which is often the case if the client is a workstation, or have an additional private disk, which is often the case if the client is a PC.

The centralised file server may easily be a bottleneck. To reduce this bottleneck more file servers are often added. The network administrators have to inspect network traffic and move disk contents between servers to reduce the load further.

The PC/laboratory instrument concept.

A concept often used in laboratories is to use a PC to control of a number of laboratory instruments, sensors/detectors etc. The PC is used to set initial values and for control of the experiment. Measured values are forwarded to the PC for further processing. Most network traffic will be between the PC and the remaining equipment.

5.2 LAN-based systems.

In the following, system research on LANs is divided into the following three categories:
• **Parallel processing:** Dense parallel processing system of homogeneous computing resources,
• **Heterogeneous multicomputers:** Computing system for large tasks consisting of general and specialised computing resources,
• **High performance LAN:** Computing system for individual tasks running on general and specialised computing engines and with a global operating system improving total utilisation of resources.
Although the main target for the SWIPP project is heterogenous multicomputers, parallel processors and LANs may also use the SWIPP hardware.

5.2.1 Parallel processing.

Parallel processing is widely viewed as a promising way to solve large calculation tasks in, for the human observer, shorter time. Subdivision of the large sequential task into several smaller tasks running simultaneously on several processors has its limitations. For all tasks there is a minimum "grain size" from which further division is not advantageous. A sub-task is less than the grain-size if the added time for preparation, communication and complementary work is longer than the time added if the application processor executed the sub-task itself. Small minimum grain size allows the subtasks to "float" more easily around to keep all computing elements busy. The strong dependence of parallel processing performance on communication time has resulted in a number of specialised MPPs (5.1.2 in this thesis). The high development pressure on general

processors and the time to design MPPs result in that new MPP systems are often based on older processors than what new single processor systems can offer. In [3] the delay is estimated to approximately one and a half years. In processor development this is significant time. Thus general LANs with new processors have a processor technology advantage over "new" MPPs.

5.2.2 Heterogeneous multicomputers.

Heterogeneous multicomputers are more an additional opportunity than an alternative to homogeneous parallel processing systems. Heterogeneous multicomputers can benefit from combination of different computing resources. The nature of many processing tasks would benefit more from a combination of specialised different computing resources than from a cluster of homogenous computers. Such a possibility may give a better performance when different parts of a task have different characteristics. This is the case for many of the larger, most demanding tasks. One part may benefit from using a vector processor, other parts from a cluster of homogeneous parallel processors or from specialised electrical circuitry. Some parts are best performed on a general computer.

An important requirement is that the network and operating system must have a general nature. This is necessary to allow computing resources of variable kinds to be connected. This generality results in that these machines more easily can take advantage of the rapid advances in commercially-available sequential processors than the MPPs can.

5.2.3 High performance LAN.

LANs (Local Area Networks) are used to connect ordinary workstations, PCs, servers, printers etc. in one or a few buildings. The workstations and PCs are mostly used interactively by individual users. Also, an ordinary LAN would benefit from increased network performance. In addition to scaled response time, from increased bandwidth and reduced latency, a number of new features can be made possible. Some of these features offer better performance to the individual user by better utilisation of the system resources. This solution would, from a hardware point of view, look very much like the heterogeneous multicomputer. The differences are mainly in the software where the heterogeneous multicomputer has a more centralised control of task queuing and task initiation.

A further discussion of high performance LANs is given in Appendix C.

5.3 Characteristics of some basic topologies.

The basic topologies to be presented in this chapter are bus, ring, all-to-all, centralised switch and distributed switches. The SWIPP topology is a network of distributed star switches.

While this chapter contains only a brief overview of topologies, some of the appendices have deeper analysis of certain topologies and switch architectures. Appendix E has an analysis of the performance of a ring topology compared to a star switch. Appendix E also contains an analysis of the performance of a N^d switch cluster. Appendix I gives a deeper analysis of the performance of the different switch architectures mentioned in this chapter. For further study, this author would recommend the following literature: [107], [82] and [45].

5.3.1 Bus

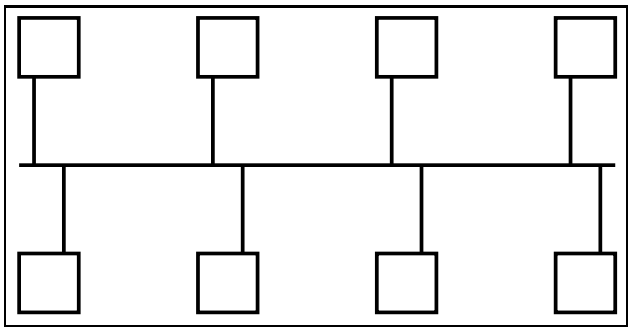


Figure 5.1: A bus topology where all nodes are connected to the same bus.

For a long time the bus (or one wired cable) has been the most popular network topology used in local area networks for connection of workstations, PCs, or laboratory instruments. It is easy to insert or take away network elements when connected equipment is to be added or removed. One of the disadvantages of the bus topology is that the bandwidth of the transport medium is shared between all connected nodes. Thus, nodes not taking part in a message exchange will temporarily lose their bandwidth when packets are transmitted between another pair of nodes. Since the average bandwidth per node shrinks inversely proportionally with the number of nodes connected, there will be a natural upper limit of nodes. This is not a problem in systems with a bottleneck at a higher level, like when all communication takes place between one node and the others (Client — Server and PC — laboratory instruments).

Another performance limiting factor in bus topologies is access control. Three access systems are used:
• A bus or wire is used for arbitration signalling. With this implementation, time may be lost for arbitration signals to propagate back and forth before the arbitration result is stabilised. This time will be proportional to the bus length. If arbitration is overlapped by data transmission, the time to forward a packet may not increase[2]. In this case the additional delay is the maximum of the waiting time for service of other queued packets and the arbitration time. (If the waiting time is longer, the arbitration time will not influence. If the waiting time is shorter (like when the transmission medium is idle), the packet has to wait for the arbitration to take place.)

• A *permission to send* token is circulated between net nodes. Some time will be lost while the token is forwarded to idle net nodes. That time will be proportional to the number of net nodes ([107] pg. 148).
• A dummy signal is transmitted in front of the real packet. The dummy signal is used to tell other net nodes to stay away from the net. The time from the beginning of the dummy signal to the beginning of the packet signal is equal to the time for the signal to propagate a wire from end to end ([107] pg. 127 and pg. 141).
All these arbitration schemes result in that some time is lost before transmission can start. This both increases latency and reduces efficient utilisation of the bandwidth.

Another factor is electrical mismatching of characteristic impedance. Impedance mismatch results in increased error rate and has to be taken care of as bandwidth increases, the buses get longer, and the number of nodes connected to each bus increases.[3]

[2] In this case this arbitration time will not influence the efficient bandwidth (4.3).
[3] In standardized networks, regulations are given on what and how components can be connected. These

Typical bus bandwidths are up to 20 Mbit/second.

5.3.2 Ring

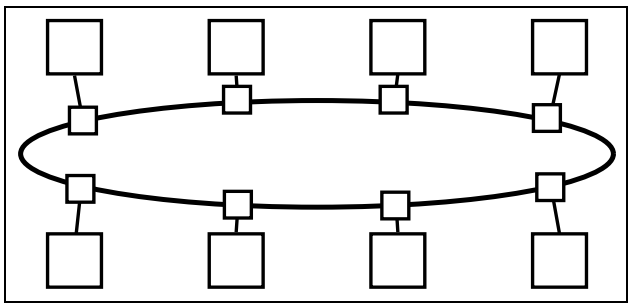


Figure 5.2: A ring topology where all nodes are connected to the same ring.

The ring topologies offer some advantages over bus structures. Each ring segment has only one transmitter and one receiver. Thus, the access arbitration can be performed locally inside each attached net node. No time is lost to propagate arbitration signals to other parts of the network. This increases bandwidth per wire and allows better utilisation of bandwidth and a larger ring. The ring structure has much of the same elasticity as buses in dynamic networks. Physically the ring topology can be implemented as cables with two wires, one for each direction. This leads to higher reliability (e.g. FDDI). As with buses, ring segments can be inserted or taken away as connected equipment is added or removed. The ring also shares a disadvantage with buses in that the bandwidth is shared between all connected nodes. Thus, to compensate for this, the bandwidth should be larger than for topologies where each link takes care of a smaller part of the traffic (This is discussed in appendix E). Net nodes not taking part in communication have temporarily no bandwidth when others have network traffic.

Ring bandwidths are in the range up to 8000Mbps (SCI [48]).

5.3.3 All-to-all and centralised switch networks.

All-to-all networks and centralised switches are topologies where link capacities are private. Thus there will be no conflicts and no waiting time for other connections to release bandwidth.

In all-to-all networks there is a private channel between each pair of net nodes. This is a network without conflicts. The total bandwidth is immediately available for the transmitter. This topology may be usable for small networks, but has mainly theoretical interest for medium and larger networks. It would be too complicated to stretch a wire between each pair of instruments in a laboratory or between computers in different rooms. The net nodes themselves would probably also be bottlenecks. They could most likely treat only a limited incoming data stream in a time unit. If they should not be bottlenecks, their capacities have to scale when the network grows. An all-to-all network would be an overkill with too many wires and too large capacity.

In central switching, all outgoing channels from a computing engine are merged into one link to a central switch. Similarly, all incoming traffic to a computing engine is routed through one

regulations put an upper limit to the expected error rate. To give these regulations, the components have to be characterized through measurements.

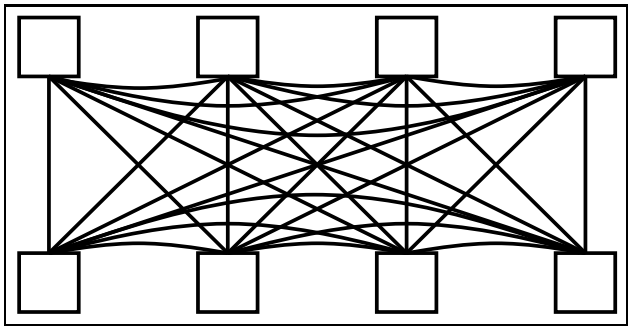


Figure 5.3: A topology with a private connection between all pairs of nodes.

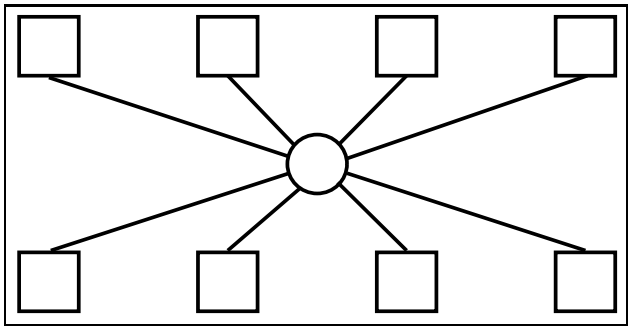


Figure 5.4: A central switch with one connection for each of the connected nodes.

private link from the centralised switch. Most computing engines can handle incoming data from only one source at a time. With centralised switching, one channel is required for each new computing engine added. Long lines and parallel wiring for distant hosts are needed. The number of connected computing engines would be limited by the central switch.

5.3.4 Distributed switches.

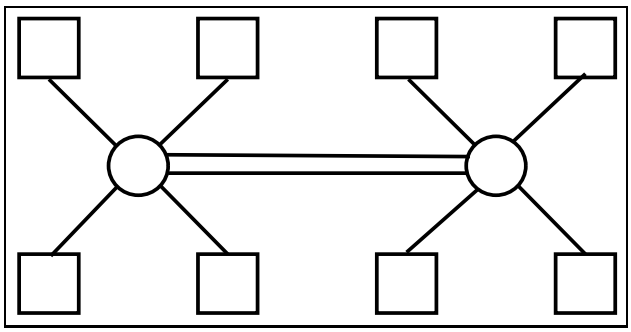


Figure 5.5: A topology with more than one switch.

In distributed switch networks switch links are either connected to computing engines (or more exact: their Protocol Engine) or to other switches. The switches may be connected in a regular

structure like a 2-dimensional, n-dimensional or tree shaped structure, or they may have any irregular structure.

Distributed switches offer a semi-private transport medium i.e. the topology and routing algorithm may decide that a link is shared between any number of connections.

For very small networks, switches and channels can be placed so that each pair of CE's has its own private channel. For larger networks, knowledge or estimates of network traffic can be used to place switches and channels for optimal network performance.

Distributed switches can offer links shared with a smaller number of connection pairs than bus and ring topologies. Thus each communication pair may have a larger part of the bandwidth and less channel contention can be achieved. Distributed switches also offer a dynamic topology where new network segments can easily be added or removed.

Compared to other topologies, networks with distributed switches have a higher capacity to utilise knowledge of network traffic to achieve higher performance. This requires more effort in topology design than the other topologies. On the other hand this work will pay in increased network performance.

5.4 Switch architectures

SWIPP has an input buffered switch architecture with simple flow control. The switching element is a non-blocking crossbar matrix. In the following, this architecture will be compared to some other switch architectures which may be used in distributed switches.

The level of independence within the network may result in that several sources may request for the same resource at the same time. Normally one source will be chosen as the winner and have the resource. The losers may be treated in one of two ways: They may be dismissed, or their request may be queued for later service. In the following we assume the last alternative. The independence of the sources result in a minimum collicition rate and a minimum queueing time. When the traffic is given, the only way to reduce the collision rate or quening time further, is through increasing the capacity of the resources.

If the system is a switch architecture, we may consider the input channels as the sources and the output channels as the resources[4]. Hence, when the input traffic pattern is given and the output channel bandwidth fixed, the minimum waiting time is given. Internal structures can not reduce the waiting time below this "externally" given limit. In an ideal switch, competition for internal resources, like buffers and local links, is minimised so that the switch performance is equal to the external, architecture-independent limit. A model for the externally decided parameters are discussed in Appendix I.

A number of different factors influence the performance of a switch. The local switch fabric may be a blocking or a non-blocking type. A blocking type is one where connections between input and an output channel pairs have to fight for the same line segments. The loser has to wait for the winner to finish the traffic. In non-blocking switch fabrics, communication between different input and output channels can take place independently of other pairs. Often the size of non-blocking switch architectures increases more than linearly with the number of input and output channels. Hence they are not practical for switches with a large number of channels.

[4] We may also consider the entire network as the system where input channels are the sources and output channels the resources.

An example is crossbar switch matrixes where the complexity increases with N^2 where N is the number of input / output channels. For larger switches a blocking network may be preferred due to their lower complexity. An example is Banyan networks, which are blocking but grow only with $N \log_2 N$. In the following, we will take for granted that the switches are small so that non-blocking crossbar switch fabrics can be used.

To reduce performance degradation due to packet loss, network switches need buffers. (The only networks where switches without buffers may be acceptable are very small networks with low load.) The buffers are used for temporary storage when outputs are busy. This is to reduce the packet loss rate. Packet loss results in retransmission which increases latency and reduces efficient utilisation of channel bandwidth. To reduce the packet loss rate, flow control is often used to secure buffers from being overflowed. Figure 5.6 shows the loss rate of a system example without flow control. Flow control is discussed further in 5.5.

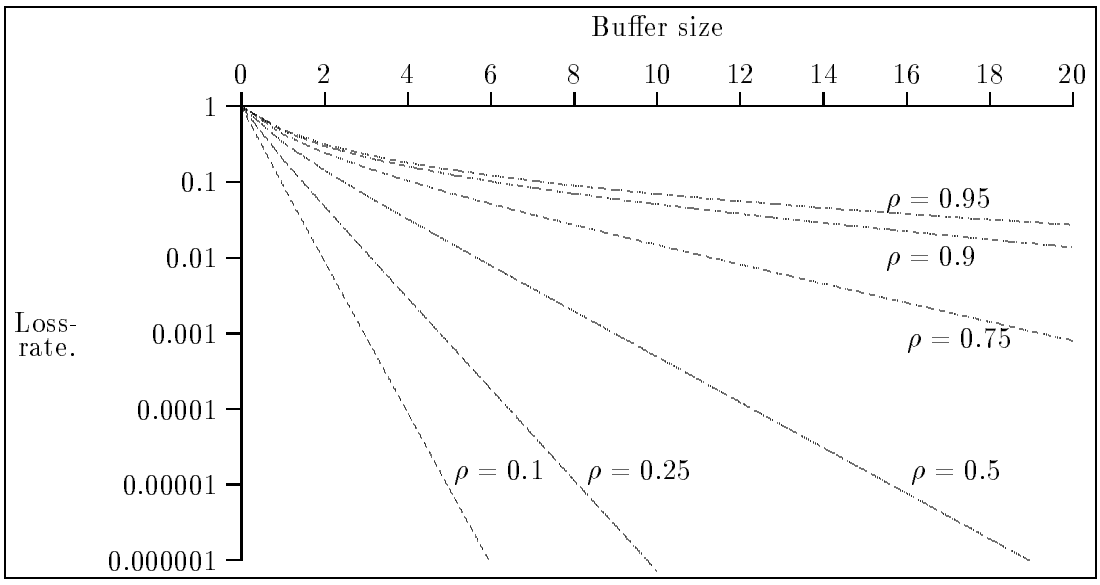


Figure 5.6: The figure shows the dismiss rate of a M/M/1 buffer without flow control. The curves show different load levels. The x axis is the buffer size in average message lengths. Note that a segmentation of a message into several packets does not change the buffer requirements if the packets are transmitted together. The buffer requirements are high when the load is high or unpredictable and messages long. (Simply explained M/M/1 is independent arrival of packets with variable length. Shorter packets are more probable than longer packets.)

The position of the buffers inside the switch will influence the switch performance. Most commonly, buffers are placed at the switch input in front of the switch matrix, at the switch output behind the switch matrix or in the center with a switch matrix on both sides.

In the input buffered and output buffered switches the buffer may be divided into several channel dedicated parts. Each such buffer part is shared between all connections using the channel the buffer part belongs to. With central buffer architecture, the buffer may be shared by all connections. The advantage of these shared buffer architectures is the relative large amount of buffer available for all connections. The clear disadvantage is that a heavily loaded, blocked

connection may occupy the buffer and block all other connections using the same channel. The present version of the SWIPP switch is using such a shared input buffered architecture.

Some switch architectures have been designed with the purpose of reducing or avoiding the neighbour-blocking mentioned in the previous paragraph. In these switch architectures buffer segments are reserved for a single connection or for a limited group of connections. This gives a significantly better performance when some connections are heavily loaded. The main disadvantages of these architectures with buffer segmentation are:
- More complex flow control.
- Lower maximum bandwidth between net nodes.
- Shorter maximum distance between net nodes.
- For some architectures: Limitation on the number of connections supported.
- Hardware-bound packet size.

This is discussed further in 5.7. (Example [55]). The blocking of neighbour connections through the same channel is a serious problem which should be investigated for further versions of the SWIPP switch.

(For references in this large area, the reader is referred to the literature mentioned in the beginning of this chapter.)

5.4.1 Input buffered switches.

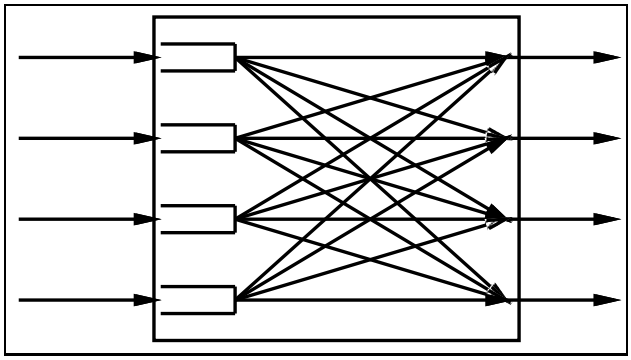


Figure 5.7: Switch with buffers at the input side and the switching matrix on the output side.

The buffers are dedicated to each input channel and are placed between the input channels and the switch matrix. This is the simplest switch architecture, and the smallest switches are generally designed with this architecture.

The input buffered switch architecture can relatively easily be designed for variable packet lengths. For centrally buffered and output buffered architectures, fixed packet lengths are easy to implement while variable packet lengths are more difficult.

Bandwidth saturation due to Head-Of-Line blockage.

Switches may suffer from a saturation degradation known as Head-Of-Line (HOL). HOL-blocking occurs when a packet routed to an idle channel is blocked behind a packet routed to an occupied output channel.

The conditions for HOL blocking are:
• *Strict buffer:. Here, "strict" buffer means that the packet forwarding sequence can not be changed depending on feedback from requested resources (channels). Thus, FIFOs and LIFOs are examples of strict buffers.*
• *Spread destinations: The packets in the buffer may request different resources (like a number of different output channels).*
• *Competion: Other channels may request for the same resources.*
• *Local flow control: The requested resources may give negative flow control signals resulting in that the requesting packet is not forwarded.*

If another input channel is occupying the requested output channel, we have HOL-blocking only if the losing buffer is notified. If it is not notified, the packet is forwarded, and dismissed at the output channel. Thus we have a packet loss and not a blocking. Flow control between switches is not required for HOL blocking. If the output channels are notifying the input channels of the same switch while switches are not notifying their predecessors, we have both packet loss and HOL blocking. [5]

There are several ways to reduce saturation degradation due to HOL-blocking:
• Reduce the number of input channels competing and the number of possible output channels chosen.
• Increase the bandwidth.
• Reduce the traffic.
• Have a structure where spreading to different destinations and merging from differnt sources take place in different positions.
• Allow some bypassing of blocked packets in the input buffer.
With an input buffered switch of this type with infinitely many input and output channels, and where all input channels choose between the output channels with equal probability, the maximum utilisation will be 58%[6] (or $2 - \sqrt{2}$) ([82] eq. 4.18) of the total switch bandwidth. If the number of input and output channels is decreased the utilisation level will be higher (Variable packet length: Table I.2 in appendix I of this thesis. Fixed packet length: [42] Table I. pg. 1590). The difference in saturation between a switch with HOL and a switch without HOL may be smaller in typical data streams when input channels choose some output channels more often than others. The utilisation level of a HOL switch may also increase if the input channels have private and different preferred output channels.[7] In the extreme case, when each input channel is transmitting to its own private output channel, the maximum bandwidth utilisation will be 100 %.

The Head-Of-Line blockage may be reduced through a "spread-buffer-compete" structure. In this structure, packets are first routed to different buffers depending on their destination. The goal is that the buffers are ready to receive packets from the input channels at once. The input channels must have private access to their buffers or share them with few others[8].The output channels will read packets from buffers which only contains packets for that output channel[9]. This structure

[5] Thus in input buffered switches, flow control between switches is not required for a switch to suffer from HOL. In output buffered switches (to be explained in the following pages), the requested resources are in the next switch. Without flow control between the switches, output buffered switches can not suffer from HOL-blocking.

[6] This is for fixed packet lengths.

[7] These assumptions are based on a limited set of unpublished simulations done by the author. No papers have been found on this topic. All of the simulated destination distributions supported the assumptions presented here.

[8] Hence the "competition" requirement for the HOL definition is not satisfied

[9] The HOL-blocking would be eliminated completely if each input channel had a private buffer of infinite length for each output channel.

may be implemented in the architecture of a switch ([64]). It may also be implemented through the connection pattern of the input buffered switches, such as the SWIPP switch. Figure 5.8 shows a topology example with three switches. Another possibility is to utilise switch connections not used. If, as default, physically unconnected switch ports have local loops back, the unconnected ports may be utilised as in figure 5.9.[10]

An efficient way to increase the traffic level at which saturation occurs, is to let packets for idle resources bypass packets for busy resources. Appendix I section I.4 contains some simulations on limited bypass. For the traffic pattern used in I.4 (i.e. uniform random), the traffic level is increased from 52 % to 68 % by allowing one bypass. Allowing three bypasses increases it to 76 %, seven bypasses gives 85 % , 15 bypasses gives 92 % and 31 bypasses gives 94 %. Thus, allowing only a few bypasses gives a significant improvement of saturation. The improvement is largest for the first packets allowed bypassed. With the traffic pattern in I.4, bypass gives less improvement below saturation. Below 30 % load, bypass gives no significant improvement. Examples of how partial bypass can be implemented are:
• One large input buffer containing several packets. This requires a multiple output buffer. Restrictions on packet length will simplify architecture.
• Multiple "logical" lanes [22]. A part of a packet is present in the switch while the remaining is stored in buffers upstream. This requires regulations on packet size, and also requires additional information in flow control signals and the packet parts forwarded.

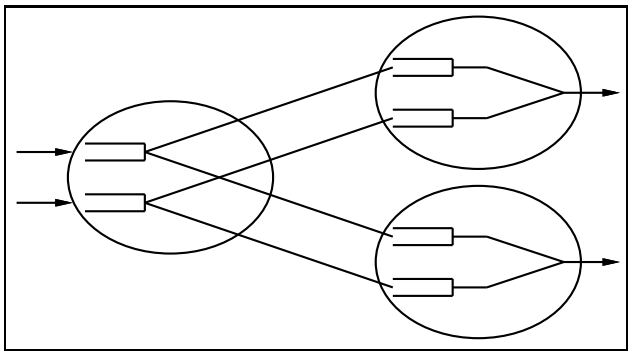


Figure 5.8: Switches suffering from HOL blockage, (both input buffered, central buffered and output buffered switches), all may be connected in topolgies giving reduced HOL blockage. This figure shows three input buffered switches connected in this way. Efficient buffer lengths have to be longer than the packets. If network load increases, the buffer length / packet length relation also has to increase to avoid an increase in HOL blockage. Thus, for high load it may be efficient that the transmitting Protocol Engine divides messages into smaller packets.

5.4.2 Output buffered switches.

The group of switches normally named output buffered are characterised by having a larger internal bandwidth between the input channels and the buffers than between switches. This increased local bandwidth may be implemented by allowing all input channels to forward packets simultaneously to the output area. The output buffers change this variable local high bandwidth

[10]Proposal of the author. No referances have been searched for and no simulations have been done.

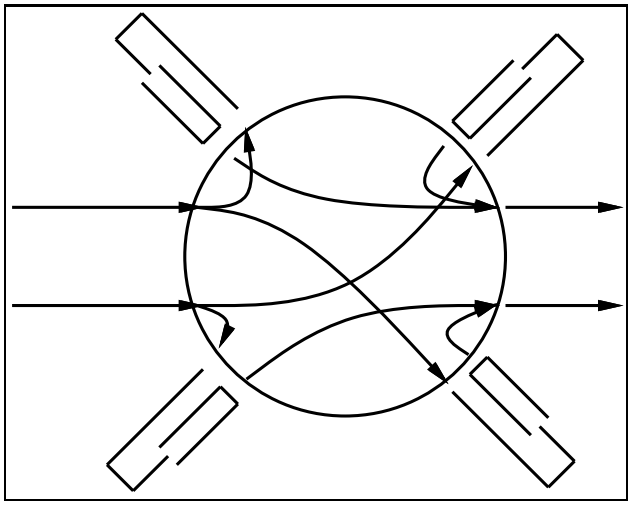


Figure 5.9: Utilisation of the unconnected ports of a switch to reduce the HOL blockage. Two input channels are accessing two output channels with similar probability. To reduce the probability for HOL blockage the packets are routed via unused channels. In SWIPP, physically unconnected channel inputs/outputs have as default an internal loop-back. The comments about buffer length / packet length relations in the previous figure text, are also valid here. This solution also has the advantage that unused channels can be connected in chains to increase the buffer lengths. Hence larger packets and/or higher load may be serviced and still the HOL blockage may be kept infrequent

stream into a more smooth low bandwidth packet stream for the channel between the switches.

An output buffered switch with n input and n output channels requires a switch fabric bandwidth n times the bandwidth of a corresponding input buffered switch.

While the switch matrix is made more complex, the arbitration logic can be made simpler, compared to the arbitration logic of the input buffered switch.

Most output buffered switch architectures have been designed for fixed packet lengths. Since variable packet lengths are more difficult to implement in hardware, message splitting into fixed packet lengths at the transmitting node is preferred.

Output buffered switches (as defined above) have less HOL blockage than input buffered switches. The reason for this is that two of the conditions for HOL,"competition" and "spread destinations" take place at different positions. Between the inputs and the buffers there are "spread destiations", but little or no competition. On the output of the buffers there is competition but no "spread destinations". The buffers give an elastic isolation between these positions. If the buffers are very short, these points will be virtually the same point, and the probability of HOL blockage will increase considerably. If the length of the buffers is increased, the probability of HOL blockage will be reduced. HOL will occur when a buffer is full. The behaviour of the output buffer can be modelled as the $M/D/1$ - system known in queuing theory [61].

There are two architecture alternatives for handling full output buffers in an output buffered switch:

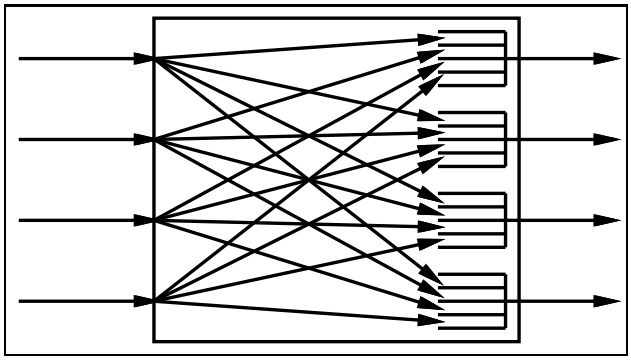


Figure 5.10: Switch with buffers on the output side and the switching matrix on the input side.

• Overflowing packets are dismissed (fig. 4.8, [45]). These types of architecture normally do not use flow control between switches.
• Additional "strict"[11] buffers at the input store packets until they can be forwarded to the output buffers. Flow control is used upstream between switches to prevent the input buffers from being overfilled. (I.e. HOL is preferred over packet loss). ([1]).

An example of an output buffered switch implemented on a circuit board is the Athena switch [83]. A variant more suited for high numbers of input and output channels, is the Knockout switch ([111] and 6.9.1 of this thesis).

5.4.3 Centrally buffered switches.

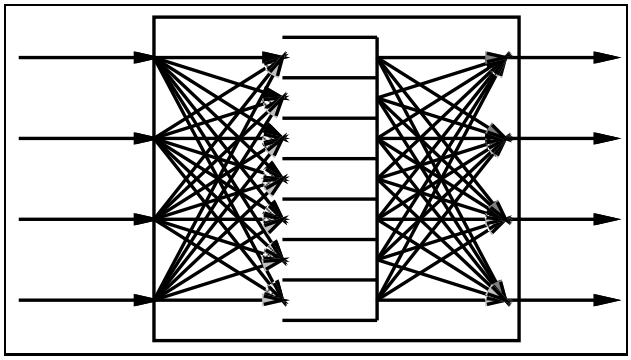


Figure 5.11: Switch with one central buffer. The switch has a switching matrix both on the input side and on the output side.

The variations in buffer usage on the individual channels are smoothed out a little by putting the buffers together in one common buffer bank. Thus a more efficient utilisation of the buffer space is achieved. Limitations on how much buffer space a connection can use, should be practised.

The buffers of most modern switches of this type can receive data from several channels and transmit data to several channels simultaneously. They also support cut-through routing i.e.

[11] Buffers not allowing packet bypassing depending on available output buffers /channels.

incoming data are forwarded to the output channel as they arrive [59]. They are most easily implemented with fixed packet lengths.

Also the centrally buffered switch may suffer from HOL blockage. To avoid this it must be possible to bypass the first packets if they are routed to busy channels. This requires both more information in the flow control signals and that the switch circuit is designed for this.

An example of a centrally buffered switch is Roxanne [39] from Alcatel.

5.5 Flow control strategies and minimum buffer sizes.

This section compares the SWIPP flow control strategy and the SWIPP minimum buffer requirements to some other strategies.

5.5.1 General: Flow control, minimum buffer size and efficient bandwidth.

Buffer sizes and flow control signalling are influenced by the product of the channel bandwidth C and the flow control reaction time t_{fcrt}. The flow control reaction time is the time from a flow control signal is transmitted upstream until the arriving data stream has changed due to the transmitted flow control signal. Thus, t_{fcrt} is the sum of the time for a flow control signal to propagate from one buffer to the closest buffer upstream[12] and the time for data to propagate from the previous to the present buffer downstream. Since this sum depends on propagation time along wires/fibres between switches, it may have different values between different net nodes. When expressed as a function of wire/coax parameters we have the inequality:

$$t_{fcrt} \geq 2l_{wire}/v_{wire} \tag{5.1}$$

Here l_{wire} is the length of the wire between the switches, while v_{wire} is the propagation velocity along the wire. When the switch circuitry is fast and the distance between the switches large, t_{fcrt} will approach the right side of eq. 5.1. To support full utilisation of channel bandwidth, a source must have the permission[13] to transmit at least a data amount of Ct_{fcrt} in each time t_{fcrt}. If the source has permission for a smaller amount of data, the transmission rate will be lower than C, i.e. the channel bandwidth is not fully utilised. This is valid for all switch architectures using flow control.

To support full channel utilisation and be capable of storing all arriving data without packet loss, the minimum buffer capacity (free and occupied) for each input channel must be equal to or larger than Ct_{fcrt} for each incoming channel.[14]

$$B_{Min} \geq Ct_{fcrt} \tag{5.2}$$

[12] In SWIPP: to the closest Input Port

[13] I.e. receive positive flow control acknowledgement signals.

[14] This inequality has a general nature for flow control systems. The collected buffer space of all net nodes, which has to be available for a connection is $B_{total-available-for-connection} \geq 2l_{total-wire-length}C_{channel-bandwidth}/v_{wire-velocity}$. For a switch we have the inequality: $B_{total-for-switch} \geq \sum_i 2C_{channel-bandwidth}l_{distance-to-neighbours}/v_{wire-velocity}$. Here, $v_{wire-velocity}$ is the propagation velocity of the signals along the wires (fibre/coaxial cable/twisted pair/e.t.c.) between net nodes. $C_{channel-bandwidth}$ is the bandwidth available between net nodes when transmitting. $l_{total-wire-length}$ is the total length of the wires between the traffic source and the traffic destination. $l_{distance-to-neighbours}$ is the wire distance between the switch for which the calculation is done and its neighbours. On average $l_{total-wire-length}$ is $l_{distance-to-neighbours}$ times the number of switch jumps.

Exactly how much larger than Ct_{fcrt} the minimum buffer size B_{Min} has to be, depends on the flow control strategy and packet sizes as discussed in the following. If the available buffer space B is too small, we have:

$$C_{starved} = \frac{B}{t_{fcrt}} \text{ when } B < Ct_{fcrt} \tag{5.3}$$

General: The bandwidth influence on buffer size.

As will be discussed in the remaining part of 5.5 the *minimum* buffer size depends on the flow control strategy. The average buffer size used depends on channel load. A doubling of the channel bandwidth will demand approximately a doubling of the minimum buffer size. On the other hand, the average buffer size used will be less than half of what it was prior to the doubling of the channel bandwidth, presumed the same network traffic. This comes from that the probability for channel conflicts is reduced so that the waiting time is less than a half. This is visualised in figure 5.12.

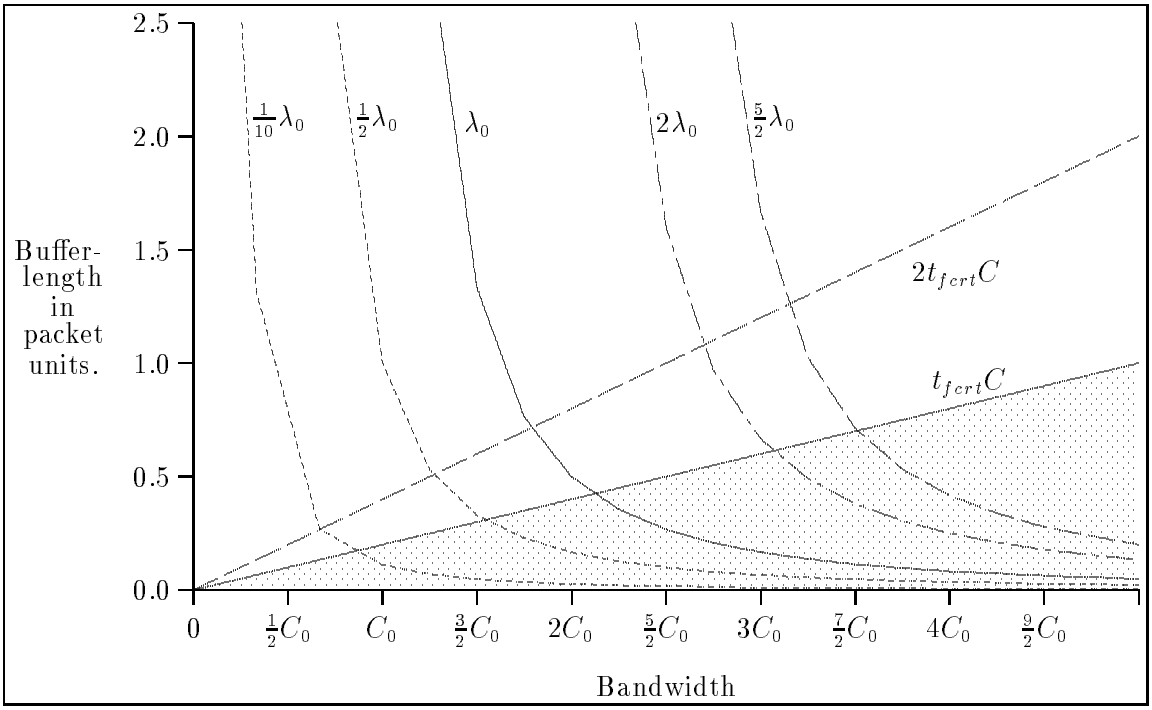


Figure 5.12: The figure shows minimum buffer requirements and average buffer utilisation as a function of bandwidth. The increasing lines are the minimum buffer requirements. The ratio of the lines, arbitrarily chosen on the figure, depends on the flow control reaction time. The lower line is the absolute minimum for any flow control system, while the upper line is the minimum for the SWIPP flow control system. The falling lines give the average buffer utilisation for some traffic levels. C_0 is a reference bandwidth. λ_0 is the traffic level, which if arrived from one source, would exactly fill the bandwidth C_0. However, since the traffic arrives from several sources the queue grows to infinity at this bandwidth.

5.5.2 The SWIPP flow control system.

SWIPP is using an implicit flow control system. Basically positive and negative flow control signals are only forwarded upstream when the flow control status is changing, i.e. the data stream is going to be stopped or restarted. The signals are inserted at once. Flow control signals are also inserted into unused space in control symbols but this additional feature has a limited influence on behaviour.[15] The SWIPP hardware supports any packet size.

In flow control systems such as the SWIPP system, the minimum buffer size for efficient communication[16] is $2Ct_{fcrt}$. The reason for this minimum size is as follows: At least a buffer space of Ct_{fcrt} has to be unoccupied when a "stop" is transmitted upstream. This is to secure storage capacity until the incoming data stream actually stops. Also an amount Ct_{fcrt} of the buffer space has to be full at the transmission of the "stop" signal. This is to secure enough data to occupy the output after a restart. More data, if available, from the previous buffer upstream will arrive after a time t_{fcrt}.

This will be explained in more detail in *12 Flow control and input buffering.*

In the following we will compare this flow control method to some alternatives.

5.5.3 Fixed interval between flow control signalling.

Another alternative could be to have positions in the symbol stream, with fixed intervals, where the flow control information is inserted. When the flow control status changes, the new value is inserted into the next passing flow control position.[17]

This method has two disadvantages compared to the SWIPP method:
a) The minimum size for the receiving Input Port buffer has to be larger. This comes from the longer average time before the data source is informed about a change in the flow control status. The minimum buffer size is equal to $B_{Min} = 2C(t_{fcrt} + t_{fc-int})$ where t_{fc-int} is the time between the flow control positions.[18]
b) The fixed flow control positions would give a small reduction in the available bandwidth for packet data. Since the flow control status would probably be unchanged most of the time, most positions would not contain new information and may be regarded as redundant.[19]
To reduce the effect of disadvantage b) the interval between flow control positions should be long while to reduce the effect of disadvantage a) the interval should be short.

[15] It may contribute to limit the damages if the previous flow control signal was a "stop transmission" signal which had been erupted by noise.

[16] With similar clock rates in the communicating network nodes.

[17] The present version of SWIPP has a FIFO in the switch Output Ports. With the alternative discussed here, there is no need for these FIFOs. The function of the Output Ports would be reduced to inserting a simple flow control bit into passing flow control positions. The function of these FIFOs is described in chapter *10 The Input and Output Port*

[18] To get an impression we may do a little calculation based on values from the present SWIPP architecture. With these values we find: Fixed interval between flow control signals gives a smaller minimum buffer requirement when there are eight data symbols or less between flow control symbols. This is the minimum buffer requirement for the Input and Output Port together. If the available buffer space is much larger than the minimum requirement the difference is insignificant.

[19] With an interval of eigth, the bandwidth efficiency is 88 %. Thus, if the interval should be small enough to make the fixed interval strategy preferable concerning minimum buffer requirement, the maximum efficiency achieved is 88%.

5.5.4 Flow control token for a fixed amount of data.

An alternative is to use a flow control system where "tokens" ("credits" or "tickets" in [59]) are transmitted to the data source upstream. Examples of new architctures using such a flow control system are found in -[10] (Myrinet), [26] and [98] (Digital Corp: GIGAswitch/ATM), [49] and [14] (Inmos C104), [73] (J-Machine) [11] (IWarp), [59] [60] [56] (Telegraphos) and [58] [57] (ATLAS I switch[20]). Each token allows the data source to transmit a specific amount of data D_t. The data source upstream stores (or counts) the tokens until they are used. We presume that the bandwidths between different net nodes are equal. t_{fcrt} may be different between different pairs of net nodes due to different physical distance. In the following, two alternative flow control strategies are discussed.

Token transmitted at the end of a buffer space release.

The amount of data that can be transmitted for each token is denoted D_t. First we will inspect a case where different switches may have different D_t and packets may have any length (Does not have to be an integer number of D_t). In this case the minimum buffer for full channel utilisation is :

$$B'_{Min} = ((n+1)/n) \cdot C \cdot t_{fcrt} = (n+1) \cdot D_t \tag{5.4}$$

The relation between n and D_t is given by the following equation:

$$n = \frac{C \cdot t_{fcrt}}{D_t} \tag{5.5}$$

We see that for increasing n the minimum buffer size in eq. 5.4 approaches Ct_{fcrt}. The disadvantage of increasing n is the larger part of the bandwidth required for the token signals. These equations are valid when tokens can not be transmitted upstream before a buffer space of D_t has been completely released.

The part of the bandwidth used for token signalling is:

$$\text{Token part of bandwidth} = \frac{M_t}{D_t + M_t} \tag{5.6}$$

where M_t is the size of the token symbol. Eq. 5.6 is also valid for the next token strategy to be described.

Token transmitted at the start of a buffer space release.

If the data are always transmitted in units of D_t, and D_t is equal for all net nodes, we can make some simplifications. Upon the start of a data read-out, a token can immediately be transmitted upstream. This is based on the knowledge that when readout of a data unit of size D_t has started, the whole data unit will always be forwarded, leaving a buffer space of D_t available for incoming data.

Also in this case the relation between n and D_t is given by eq. 5.5. The recommended minimum buffer size is smaller:

$$B''_{Min} = Ct_{fcrt} = nD_t \tag{5.7}$$

Thus, a connection with this flow control strategy must have at least n tokens.

[20] No connection with the ATLAS detector at CERN.

Tokens belonging to logical connections: "virtual channels"

Until now we have considered tokens allowing real data transmission. In some networks each physical connection is divided into several logical connections (also entitled "lanes", "paths" or "virtual channels"). (This division is performed to attain a bypass behaviour, to improve the saturation due to HOL blockage.) Private flow control signals are required for each logical connection. If each logical connection has a private memory, the flow control will signal a permission to send a data unit. If memory is shared between several connections, two flow control systems are required: One for the connections and one for the memory. In this case one token of each kind is required for data to be forwarded. Thus, one disadvantage of dividing connections into logical connections is that the efficient bandwidth, as given by the equation below, is reduced.

$$\text{Data part of bandwidth} = \frac{D_t}{D_t + M_t + VC_t + VC_{id}} \tag{5.8}$$

As in equation 5.6 M_t is the size of the memory token. VC_t is the size of the token belonging to the virtual channel. Each data element D_t has to carry an identification of which virtual channel it belongs to.

As an illustration, we may use values from the paper often referred to as the first introducing virtual channels. In [22] pg. 64 D_t is 32 bits. Sixteen virtual channels is used. Thus VC_t is 4 bits and VC_{id} is 4 bits. M_t is not used. This gives an efficient bandwidth of 80 %.

5.6 Maximum required buffer sizes.

A maximum required buffer size may be found, provided that a global flow control strategy is practised.[21] At the source, the minimum time from a data bit is transmitted until a positive acknowledgement is received from the destination, is $(2l_{connection-wire-length}/v_{wire-velocity} + t_{delay-in-switches})$. To utilise full channel bandwidth, the transmitting source has to allow an unacknowledged data amount $L_{global-fc-data-amount}$ of at least $C_{channel-bandwidth}$ times the time given above, to be fed into the network. If the amount of unacknowledged data grows above this amount, there is a blockage, and no more data should be fed into the network. In the following, we assume the sources have a global flow control strategy according to this. In the case of a blockage, we expect each source to transmit approximately the amount of data given above, before the transmission is stopped. The maximum storage capacity required if an output channel was blocked, would be the sum of the maximum unacknowledged data amounts for all connections passing the blocked output channel. By using average values we have: $N_{\text{connections}}(2l_{connection-wire-length}/v_{wire-velocity} + t_{delay-in-switches})C_{channel-bandwidth}$. We may replace $(2l_{connection-wire-length}/v_{wire-velocity} + t_{delay-in-switches})$ by $N_{\text{Average-number-of-jumps}}C_{channel-bandwidth}t_{Average-local-fcrt}$. With this replacement the total buffer used will be: $N_{\text{connections}}N_{\text{Average-number-of-jumps}}C_{channel-bandwidth}t_{Average-local-fcrt}$.
Here $N_{\text{connections}}$ is the number of source - destination connections passing the channel. $N_{\text{Average-number-of-jumps}}$ is the average number of link segments of a connection. A large buffer may be required. When the connection is not in saturation, only a small part of the buffer would be used[22]. These buffer sizes are required in single-lane architectures when a blockage should

[21] This estimation has a close connection to the calculation done in the footnote at equation 5.2.

[22] We also see that as long as the buffer has this size, we do not need local flow control, since the memory will never be overfilled. A condition for this is that the number of connections, the number of jumps and the distance from source to destination are within the limits

not be able to disturb other connections. Thus, in large networks with single lane architectures, limitation of blockage can demand large buffers. If implemented, it would imply fixed/maximum number of connections, number of jumps and physical distance. Thus it would not be scaleable.

If a host source always has an unacknowledged data amount equal to $N_{\text{Average number of jumps}} C t_{fcrt}(Average)$
also this amount would be doubled if C was doubled. Since this amount is the maximum data amount a connection can receive from a source, the "maximum" buffer space would also be doubled. A doubling of bandwidth doubles the minimum and maximum required buffer, reduces the latency and the average buffer space used. Thus a doubling of bandwidth may be regarded as requiring more unused buffer space.

5.7 Buffer segmentation for isolation of buffer blockage.

In the previous text the minimum buffer space required for each input channel was discussed. In this part we will discuss how buffer space above the minimum requirement may best be utilised. Here, no decisions have been made for the SWIPP switch. The reason for this is that the high bandwidth, the long physical distance between switches and the chosen flow control strategy all contribute to a high minimum buffer requirement. The buffer space available on one chip has been considered not to be very much above the minimum requirement. As smaller scaling and larger chips make more memory available, efficient utilisation of the "additional" buffer space is important. The preferred utilisation would be to increase bandwidth further. This will reduce transmission time and queueing time, and move the system out of saturation. When increased bandwidth is not possible, some of the solutions suggested in the following, should be considered.

One alternative is to let each of the buffer segments grow without further regulations. Then, for example for an input (output) buffered switch, the connections passing an input (output) channel would have increased their common buffer capacity. Increasing the buffer banks of the channels may make variations in buffer usage more local and reduce the probability of influencing on other switches. A disadvantage is that a blocked connection through a channel may occupy all the buffer space of a channel. Thus also otherwise open connections through the same channel may be blocked. Another possible disadvantage of increased shared buffer space is that it may take longer time for the data source to experience the blockage. Knowledge about blockage may be gained faster through a end-to-end flow control system: The source may only be allowed to transmit a data amount equal to $t_{fcrt-connection} C_{channel-bandwidth}$ (the amount found in 5.6) without receipts from the receiver. Here $t_{fcrt-connection}$ is the minimum time for a signal to propagate from source to destination and back. $C_{channel-bandwidth}$ is the bandwidth. The present version of the SWIPP switch may be considered to have this architecture.

The buffer of an input or output channel may be split into segments reserved for groups of connections using the channel to reduce the influence of a blockage on other connections passing the same channel. Segmentation at different levels will be discussed in the following. In the notation that will be introduced, the shared buffer as described in the previous, is a zero-level segmentation buffer.

Zero-level buffer reservation scheme.

Figure 5.13 shows a zero-level buffer reservation scheme. In this case a blocking of any connection through channel 1 may result in a blockage for the packet in A.

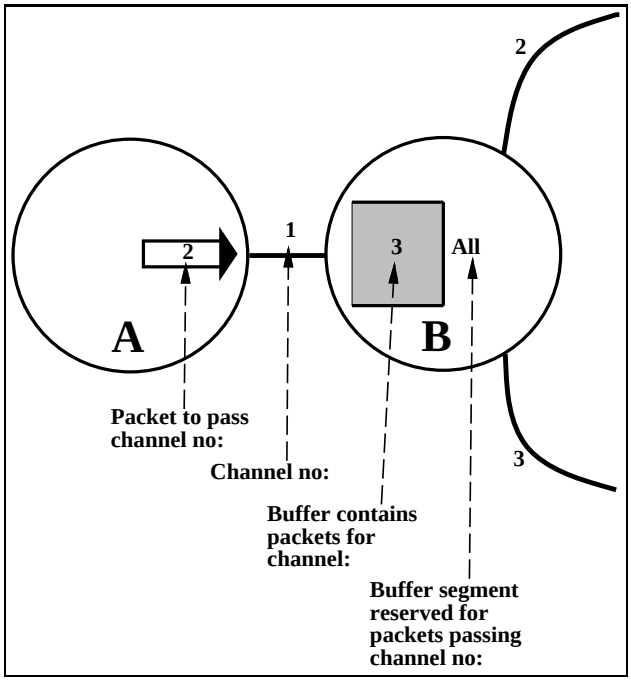


Figure 5.13: Zero-level flow control. A packet in switch A has an address path through channel 2. The buffer available for A in B is shared between all outgoing connections. Thus data for channel 3 may block the packet for channel 2 in A.

The buffer drawn in figure 5.13 is the maximum buffer space available for the channel from switch A. The buffer may be shared by all incoming channels (central buffering) or it may belong to the incoming channel from switch A alone (input buffering or output buffering). If the switches are output buffered, we may regard the figures as logical descriptions, while the true placement of the buffer drawn in B would be at the output of switch A.

If each connection should have the capability to utilise a full channel bandwidth alone (when other connections are idle), the buffer has to have a size fulfilling the $B_{Min} \geq Ct_{fcrt}$ requirement discussed in 5.5.1. In the following discussion, the buffer will contain several buffer segments. For the connections owning the buffer to have the full bandwidth possibility, the requirement $B_{Min} \geq Ct_{fcrt}$ has to be fulfilled for each buffer segment. This is an assumption and will not be repeated in the following.

One-level buffer reservation scheme.

In the one-level buffer reservation scheme in figure 5.14 there is a smaller number of connections which can block the packet in switch A.[23] The connections capable of blocking are connections through channels 4 and 5. Blockage of connections through channel 3 will not influence on the packet. To generalise, we let m be the number of outputs of B. If the buffer is divided into m segments, blockage only on $1/m$ of the connections passing channel 1 will block buffer space for the packet in switch A. Blockage on any of the remaining $(m-1)/m$ connections will not influence

[23] In the following we consider only other connections passing through channel 1. Connections entering through other than this channel may also influence on the buffer reservations. However, the point to be advocated about reduced contention with increased segmentation, is still valid.

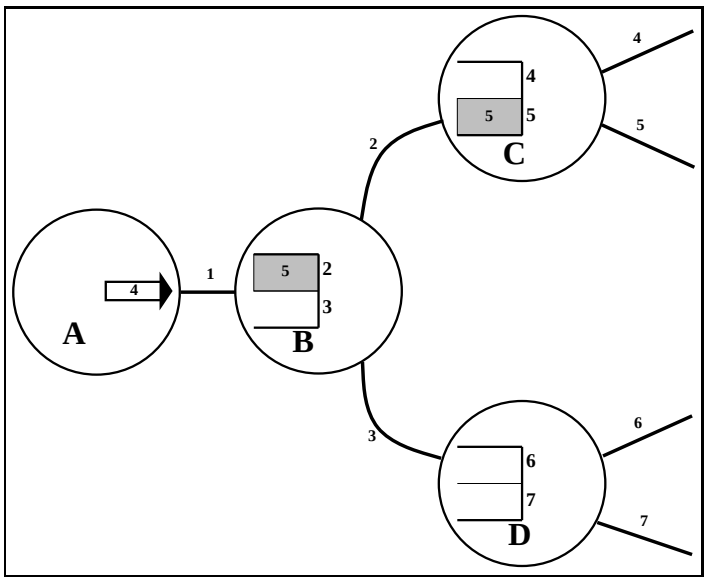


Figure 5.14: One-level flow control. A packet in switch A has an address path through channel 4. The buffer in B is divided into segments, one for each output channel.

on the packet in A.

Two-level buffer reservation scheme.

Figure 5.15 shows a two-level buffer reservation scheme. In figure 5.15 blockage on channel 5, 3, 6 or 7 will not take buffer space from the packet in switch A.

n-level buffer reservation scheme.

Generally, for a n-level buffer reservation scheme where all switches have m channels, the buffers are divided into m^n segments. Blockage on $(m^n - 1)$ of the m^n connections passing channel one, will not take buffer space from the packet in A. The required buffer space in B is m^n segments.

The conclusion is that with increasing segmentation of the buffers, blockage of some connections will spread to a smaller number of other connections. Thus a higher resistance to epidemic blockage may be achieved. Note that with this architecture, all connections will have some buffer space, if not immediately, then after some time.

In SWIPP this buffer strategy can be implemented with the present address pattern. In the case of a n-level buffer reservation scheme, the n first address nibbles will decide which buffer segment the packet belongs to. The architectural differences would be the segmentation of the buffers and that the flow control signals would belong to dedicated buffer segments. Thus the flow control signals have to carry ownership information.

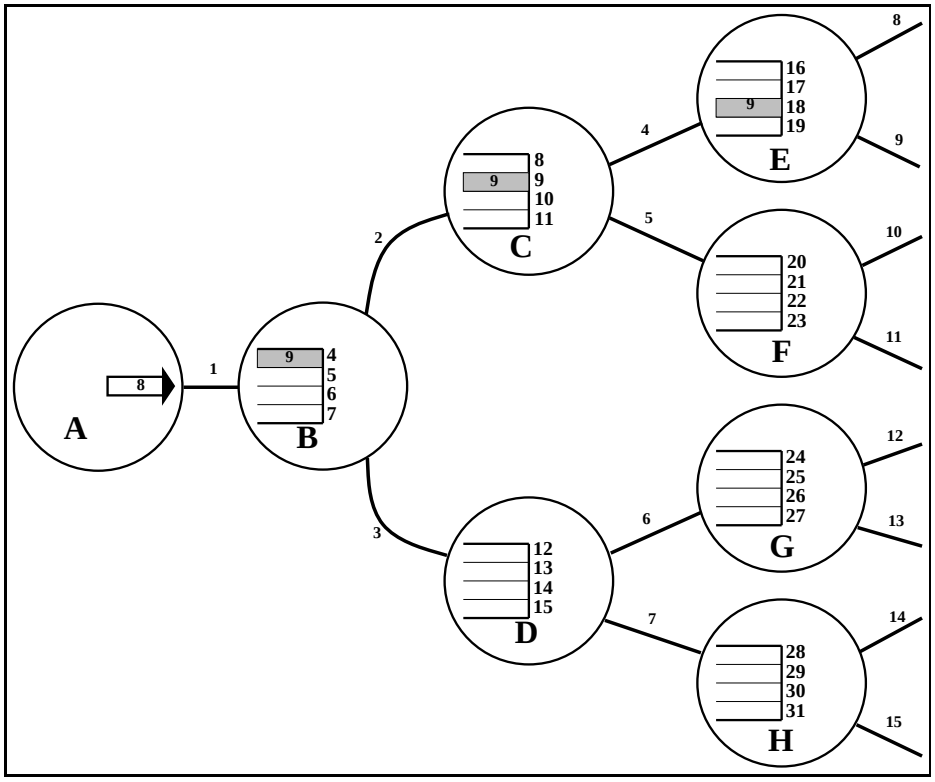


Figure 5.15: Two-level flow control. A packet in switch A has an address path through channel 8. The buffer in B is divided into segments, one for each of the output channels of the next switches.

Virtual channels.

When n in the n-level buffer reservation scheme increases, the probability of reservation of buffer segments for passive connections also increases. Then it may be more efficient to reserve the buffer segments for the connections or groups of connections in use. A buffer reservation scheme where buffer segments are reserved for individual connections, is denoted "virtual channels". One important disadvantage with this strategy is additional circuitry for initiating and updating address translation tables. Another disadvantage is that the number of connections supported will be limited by the buffer and table space. When all virtual channels are in use, no more connections can be supported. Telegraphos ([60] and [56]) is an example of a switch with one connection in each virtual channel. In the ATLAS I switch ([58] [57]) several connections are grouped together in one virtual channel.

Combined buffer reservation schemes.

Combinations of different buffer reservation schemes may be advantageous. An example is schemes which separate connections from each other as long as "virtual channels" are available, but also offers some bandwidth for all new connections when the virtual channels are occupied. Some part of the buffer may be dedicated to virtual channels, while the remaining part may be reserved for n-level buffer reservation schemes. This gives a system with two service classes where one class has connections with better protection against other connections than the other class has.

5.8 Worm-hole and store-and-forward packet strategy.

Inside a switch, two main strategies, that will be described in the following, are used for forwarding of arriving data to the switch outputs. The choice of strategy may depend on error rate and packet format. The chosen strategy influences on the packet propagation time and thus on the network performance.

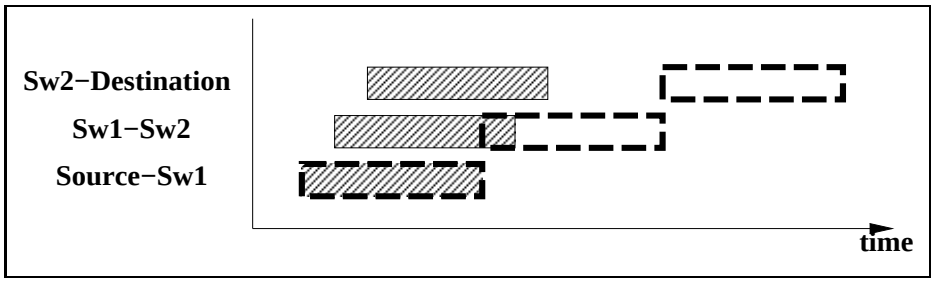


Figure 5.16: The time delay until incoming packets are forwarded influences on the total propagation time for the packet. The figure has two examples: the boxes drawn with heavily dashed borders represent a packet forwarded according to a "store-and-forward" strategy. The grey boxes symbolise a packet forwarded by a "worm-hole" strategy. In the common understanding of "worm-hole", the packet is forwarded as soon as a small data amount of a few bytes has been interpreted and the connection established. The reception of the packets at the first switch is shown in the lower row. The transmission between the first and second switches is shown in the second row, while the transmission between the second switch and the destination is drawn in the upper row.

5.8.1 Store-and-forward packet strategy.

With the store-and-forward strategy ([107] pg. 89) a whole packet is received before it is forwarded to the next switch node. The reason for collecting the entire packet may be that the connections are noisy. Thus error check with error correction or retransmission has to be performed between each pair of network nodes along a path.

Another reason may be the need to compact the packet before forwarding it. The packet may be received on a channel with low bandwidth or it may be received with idle symbols in-between. If the packet is going to be forwarded on a high-bandwidth channel or a channel with much traffic, it will be an advantage if the whole packet could be delivered together.

5.8.2 Worm-hole routing strategy.

In the worm-hole routing strategy ([91] [21] and [22]), packet parts are forwarded by each switch without error check. A low error rate is required. Only a smaller part of a packet, typically a few bytes, are collected before they are forwarded. This small data unit is often entitled a "flit" based on [21]. In [21] the authors define a flit to be "the smallest unit of information that a queue or channel can accept or refuse". In an later example in [22] one of the authors of [21] quantifies a flit to be 32 bits.

5.8.3 The SWIPP packet forwarding strategy.

The SWIPP forwarding strategy is not exactly a wormhole strategy but has more similarities than differences. Error check is not performed by the switches on passing packets. The flow control is different from wormhole since SWIPP signals start or stop on a data stream and not on individual elements. The smallest data part of a packet which can be accepted and forwarded alone, is one byte. Thus, to some extent we may consider SWIPP to have a flit size of one byte. The SWIPP routing strategy can start forwarding of a new packet sooner than ordinary worm-hole routing strategy. The only delay required is the time used to receive the destination address (the first two bytes) and to establish a connection according to the address.

5.8.4 How the choice of packet forwarding strategy influences on the transmission time.

In the following we will find an expression for the difference in propagation time between worm-hole and store-and-forward strategies. To simplify the comparison, no other traffic is assumed. The time (in general) for a packet to be transmitted through a number of switches may be expressed as:

$$t_{wire} + nt_{dec} + n\frac{L_{sw-buff}}{C} + \frac{L_{pk}}{C} \tag{5.9}$$

Here t_{wire} is the signal propagation time along the wires and t_{dec} the time for a switch to decode an address and establish a connection. The last term, L_{pk}/C, is the time the destination uses to receive the entire packet from first to last bit. The elements of the equation mentioned so far are independent of when the switch starts forwarding the received data. The difference is in the $nL_{sw-buff}/C$ element. Here n is the number of switches to be passed, $L_{sw-buff}$ the part of the packet received before forwarding can start, and C the bandwidth. With store-and-forward routing, $L_{sw-buff}$ is equal to the packet length: L_{pk}. For worm-hole routing, where the packet is divided into flits, $L_{sw-buff}$ is equal to the length of a flit: L_{flit}. In SWIPP only the two first bytes have to be received before a further forwarding can start. Thus for SWIPP $L_{sw-buff}$ is equal to $L_{dest-addr}$ where $L_{dest-addr}$ has a length of 2 bytes. Compared to wormhole, store-and-forward routing has an added delay of $n(L_{pk} - L_{flit})/C$. To illustrate this difference we may use an example. With a packet length of 200 bytes, a flit size of 32 bits, a bandwidth of 800Mbps and five switch jumps, the difference between store-and-forward and wormhole will be $9.8\mu s$.

Bandwidth relation.

If the store-and-forward strategy is chosen due to high error rates, it may be worth considering to reduce the bandwidth and thus also the error rates so that worm-hole can be chosen instead. Let $C_{worm-hole}$ be the highest bandwidth with an acceptable error rate for worm-hole routing. For store-and-forward to offer a shorter latency (i.e. propagation time and packet length/bandwidth time), the following condition has to be fulfilled:

$$C_{store-and-forward} > \frac{nL_{pk} + L_{pk}}{nL_{dest-addr} + L_{pk}} C_{worm-hole} \tag{5.10}$$

When n is large, equation 5.10 may be simplified to:

$$C_{store-and-forward} > \frac{L_{pk}}{L_{dest-addr}} C_{worm-hole} \tag{5.11}$$

A $C_{store-and-forward}$ satisfying eq. 5.11 will always satisfy eq. 5.10.

The general conclusion in the choice of routing strategy for switches, is that for the store-and-forward strategy to have shorter transmission time than worm-hole, the network must have a bandwidth given by eq. 5.11.

5.9 Comparison between fixed and variable packet size.

The advantage of a packet format allowing variable packet length is a more concentrated data stream giving more efficient utilisation of bandwidth. If a large data message is divided into packets with fixed lengths, packet heads and packet delimiters will be required. In the following we will do some calculations on the utilisation of bandwidth when a message is segmented into several fixed-size packets.

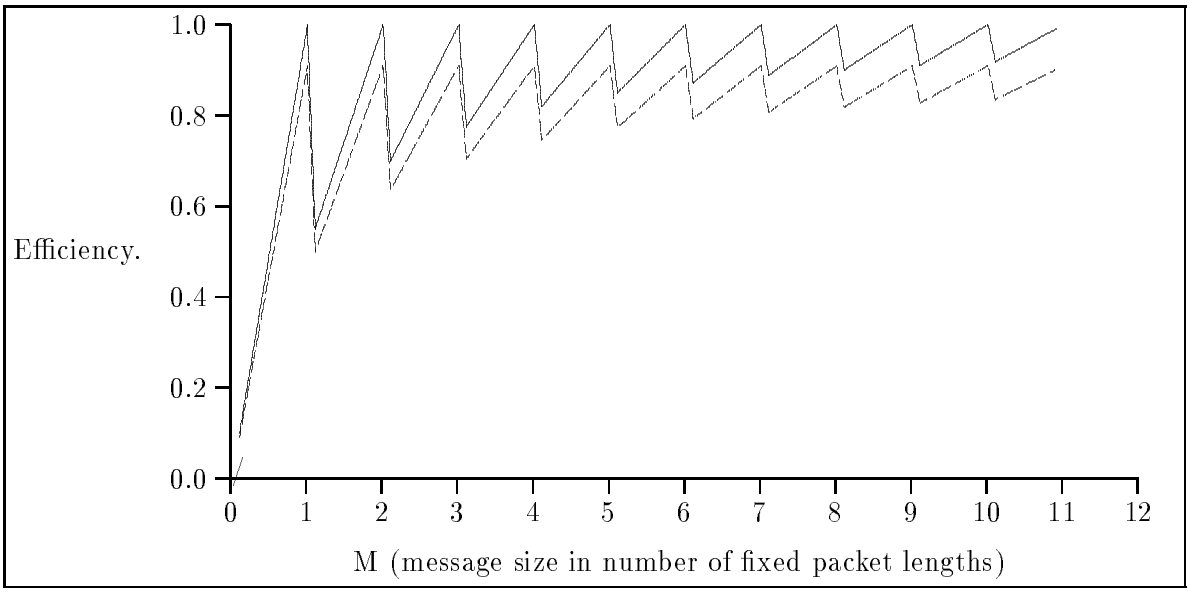


Figure 5.17: The bandwidth efficiency when a message of length M is divided into packets of fixed size. The solid line is for packets with no head. The dashed line is for packets where the length of the head is 10 % of the data part. The maximum value and the asymptotic value is: 1/(1+H/D). The minimum value is: M/(((M/D)+1)(D+H)). D is the length of the data part of a packet. H is the length of the packet header.

We let D be the amount of data in each fixed-size packet. The message length is variable with an average length M. On average the last packet will be half full with data. For fixed packet sizes the last part has to be filled with dummy data. We use the following expression between the fixed data amounts of the packets and the average message length: $M = (m + 1/2)D$. Here m is an integer. The total number of packets required is $(m + 1)$. We also introduce H as the additional packet headers and delimiters required for each packet when segmented. The transmission efficiency can be expressed as:

$$\text{Transmission efficiency} = \frac{\text{Data}}{\text{Data, headers and dummies}} = \frac{D}{D+H}\frac{m+1/2}{m+1} \tag{5.12}$$

Message length normalised on packet data length.

We introduce h as the relation between H and D: $h = H/D$. Thus the packet length is $(h+1)D$. Expressed with h, the transmission efficiency may be expressed as:

$$\text{Transmission efficiency} = \frac{1}{1+h}(1 - \frac{1/2}{(m+1)}) \tag{5.13}$$

We have the maximum transmission efficiency for $m = \infty$. At this value of m the efficiency approaches $1/(1+h)$. The minimum transmission efficiency is at $m = 0$. At $m = 0$ the transmission efficiency is half of the maximum efficiency: $(1/2)1/(1+h)$.

Message length normalised on packet header length.

In practical implementations an alternative could also be to take as basis the length of the packet header. The length of the packet header is often given from the switch architecture and the network design. Based on this length we may find the optimal packet data length when the average message length is given. By setting $H = 1$, we have that M is given relative to H.

$$E(m) = \text{Transmission efficiency} = \frac{M}{(M+m+1/2)}\frac{m+1/2}{m+1} \tag{5.14}$$

Figure 5.18 shows the data, header and dummy part of the bandwidth for different decomposition numbers.

Based on equation 5.14, we will find the optimal number of packets and a recommended fixed packet length, given an average message length M. We see that equation 5.14 has minimum efficiency levels for low and high m. We find the maximum value for m by deriving eq. 5.14 on m.

$$E'(m) = \frac{-M(m^2 + m + (\frac{1}{4} - \frac{1}{2}M))}{(M+m+1/2)^2(m+1)^2} \tag{5.15}$$

We find the m giving the maximum value of 5.14 to be:

$$m_{Max} = \frac{\sqrt{2M} - 1}{2} \tag{5.16}$$

By inserting m_{Max} in equation 5.14, we find the maximum transmission efficiency to be:

$$E(m_{Max}) = \frac{2M}{2M + \sqrt{2M}}\frac{\sqrt{2M}}{\sqrt{2M} - 1} \tag{5.17}$$

We see from table 5.1 that segmentation of messages of continuously variable lengths into fixed-length packets results in reduction of the transmission efficiency. This is especially significant for short messages.

We can illustrate the results above with an example: The switch architecture and routing strategy implies a packet head of one byte. The average message length is 1000 bytes. From table 5.1 we

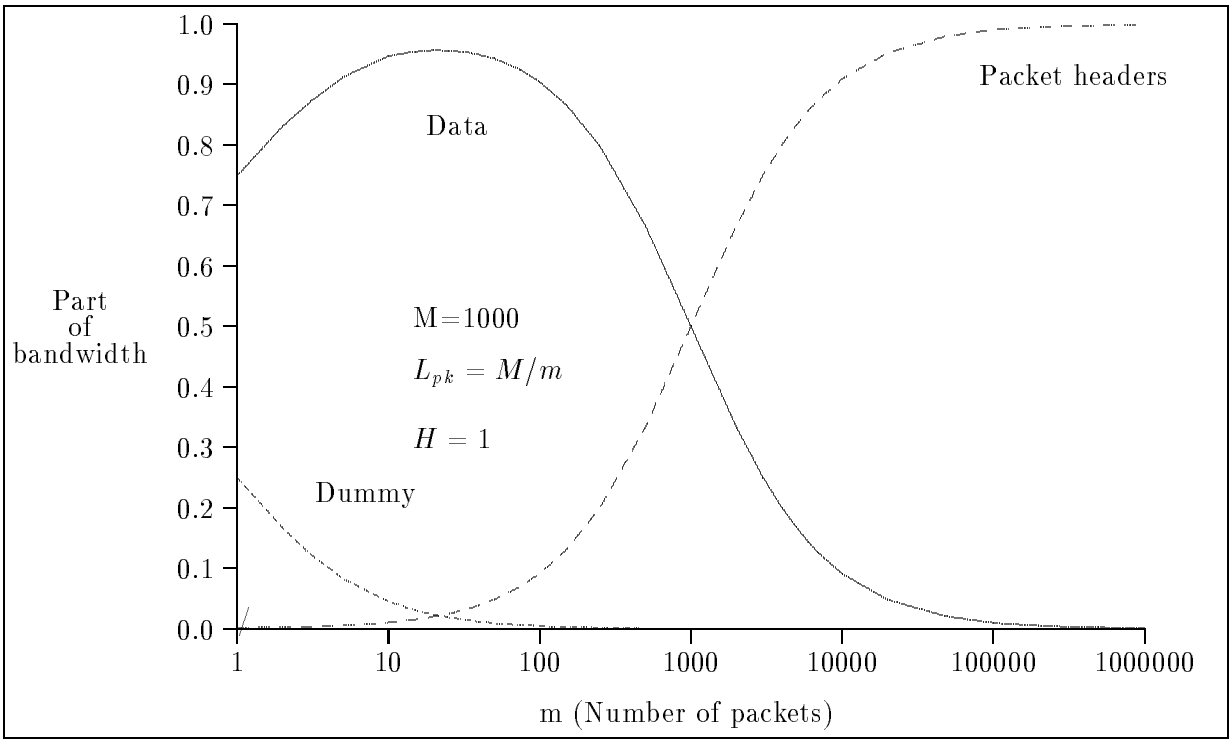


Figure 5.18: The data, header and dummy part of the used bandwidth. The messages have an average length of 1000H, with similar probability for all lengths in the neighbourhood of 1000H. The average message length M is divided into m packets of size M/(m+1/2). If the packet length instead of being fixed was a maximum value, there would not be a dummy part.

see that, to achieve the best bandwidth utilisation the fixed packet length should contain 44 data bytes. Thus the total packet length is 45 bytes. With this packet size 23 packets are required for a 1000- byte message. An important condition for this is that the message length is the average length and that the different lengths in the neighbourhood have similar probabilities, i.e. the probability that the message has 997, 998, 999,1000,1001,1002, 1003 bytes (and the other values around) is similar. If the message lengths are only ....,900, 950, 1000, 1050, 1100, 1150 ...bytes and not the values between, other considerations have to be taken, and other packet lengths may be preferred. With the conditions for the calculations above, the efficiency is approximately 96%. A result of this is that a single packet of approximately 1000 bytes occupies the channels 96% of the time an approximately 1000-byte message divided into fixed-length packets would.

Software-decided fixed or maximum packet length restrictions may be executed by the Protocol Engines. Buffers may be more easily implemented with fixed packet lengths for some of the more advanced switch architectures. Fixed packet lengths also make a smaller minimum buffer size possible, as discussed in part 5.5.2 of this chapter. The blockage behaviour with fixed and variable packet lengths will probably not be different.

Message length in H: M	Number of packets : $(m+1)$	Data length of each packet: D	Transmission efficiency:
1000	23	44	96%
100	8	13	87 %
64	6	11	84 %
8	2	4	64 %

Table 5.1: The table shows the reduction in efficiency for some long messages when they are segmented into several packets of fixed length. The reduction is relative to the efficiency of one long packet with the size of the message The messages are of variable length (continuous variation), with average length equal to M. The fixed packet lengths chosen are the optimal ones. The transmission efficiency is found through the equations above.

5.10 A simple relation between topology, bandwidth, load and latency.

In appendix E an ineqeuality was developed, which could be used for comparing two sets of load and bandwidth for M/M/1 systems. The inequality is used to find the pair with the lower latency (i.e. sum of waiting time and service time).

The inequality can be used to compare a ring and a star topology. It can also be used to illustrate the need for segmentation as networks grow. Obviously, it can also be used to find the change of transmission time for a new load and a new bandwidth compared to the present (/old) parameters. The utilisation of this equation will be illustrated in the following.

$$C_{new} > n\lambda\overline{L_{pk}} + C_{old} - \lambda\overline{L_{pk}} \tag{5.18}$$

In the inequality, the syntax has been chosen for an example with a "new" bandwidth and "new" load, comparred to an "old" bandwidth and an "old" load. C_{new} is the new bandwidth while C_{old} is the old. Both the new and the old packets have an average length of $\overline{L_{pk}}$. The old traffic level is λ. The new traffic level is n times the old: $n\lambda$. Equation 5.18 expresses the relations for which the new conditions give a shorter latency than the old conditions.

It is important to emphasise that equation 5.18 is based on a traffic model with infinitely many sources delivering packets to the network medium. The average packet number delivered to the medium per time unit is λ. Blocking, except due to other packets to the same medium, is not included. The equation is based on variable packet lengths and negligible time to propagate a single bit along the wires. For a traffic $n\lambda$ on a channel with a bandwidth C_{new} to be better than the reference for all load levels, the requirement $C_{new}/C_{old} > n$ has to be fulfilled.

In appendix E we use a variation of eq. 5.18 to compare a ring and a star topology. Now we may use the inequality to express when ring is faster than star. Thus we have to replace C_{New} with C_{ring} and C_{Old} with C_{star}. Now λ is the load contributen of each connected node. On the star the load will be λ on each of n channels. On the ring, the load is $n\lambda$ on the ring segment. Typically high bandwidth may more easily be implemented on a ring than on a star. As long as

eq. 5.18 is fulfilled, the ring will give a shorter transmission time. As the number of connected nodes increases there will be a limit to where further bandwidth increase is practical. If a star channel offers a bandwidth of $633Mbps$ and 1000 nodes are to be connected, the ring should have a bandwidth of $633Gbps$, which is clearly not practical. For such high numbers, the ring has to be divided into several subrings giving an efficient smaller n per ring which can fulfil eq. 5.18. Such subdivision is a part of the structure of the star topology and no fundamental change of the topology is necessary as the network grows.

Also when we consider splitting the channel bandwidth of one channel into two channels, we may utilise eq. 5.18. When the channel bandwidth is split into two equal halves, $C_{New}/C_{Old} = 1/2$. If the initial load was 100% and a splitting would give an advantage resulting in a small reduction in traffic, i.e. n becomes a little below 1/2, the splitting would be beneficial. If the initial load was 50%, a channel split would result in a longer transmission time, independently of possible reduction of traffic.

Chapter 6

Other multicomputer research network systems

Several research and commercial network systems having similarities with SWIPP are now being developed and are available around the world. In this chapter some of these network systems will be presented and compared to the SWIPP network system.

This chapter will present some network systems with similarities to SWIPP. They have been chosen for presentation to give an impression of the SWIPP performance or to give ideas of some possible additional functions SWIPP may be given at a later time. The networks to be presented are:
• The **Nectar** ([5], [18], [100]) research system from Carnegie Mellon University, USA, consisting of switches, communication protocols, and communication boards connected to workstations.
• The **Autonet** ([89]) research system from Digital Equipment Corporation, USA, consisting of switches and communication boards and protocols for communication between workstations.
• The **Telegraphos** (Telegraphos I - FPGA version: [55], Telegraphos II - ASIC-version: [60] [56]) research system from the Institute of Computer Science, Crete, Greece consisting of switches, communication boards and protocols for communication between workstations.
• The **Mosaic**, **ATOMIC** ([17]) and **Myrinet** ([10]) trilogy from research and commercial institutions around California Institute of Technology (Caltech), USA. The Mosaic switch and ATOMIC network system were created at Caltech and later further developed and commercialised into the Myrinet system.
An overview of these networks is given in table 6.1.

The *SCI - Scaleable Coherent Interface* [48] network system has been left out in this comparison. SCI is a protocol system with the aim to keep caches coherent in a shared memory system. The SCI cache coherent protocols do not require a specific network or topology. Anyway, a network interface often denoted as "the" SCI network interface has been developed. When several such network interfaces are connected in a chain, a ring topology of the "insertion buffer"-type is created. SCI is left out in this chapter since the focus of our application is star topologies. Other ring topologies which are interesting, but have been left out are: FDDI [13] [52] (fiber ring), CSMA/RN [29] (ring), HANGMAN from Hewlett Packard [108] (ring), PlaNET from IBM-Paris [15] [16] (Ring-shaped bus). DQDB [47] is based on two uni-directional buses. Thus it is not exactly a ring, but has a similar behaviour.

Another hot topic concerning networks is the ATM (Asynchronous Transfer Mode [82]) protocols

defined by the International Telecommunication Union and the ATM forum[1]. This ATM protocol defines packet lengths, packet formats, packet transmission (sequence and source density) and address format. The ATM standard is meant to be used for a wide range of different applications like telephone, data and video. The ATM standard of the ITU and the ATM forum may be implemented with different topologies and switch architectures. Thus the ATM protocols may, with some simplifications, be implemented with all the network systems discussed in this chapter. A number of "ATM-switches" have been implemented by industry and by universities. These switches have been designed for long distance communication, mostly telephone traffic. They often did not have flow control and thus they allowed packet loss when buffer capacities were exceeded. In the last years ATM switches targeting data and LAN have been developed by Fore, 3Com, IBM and Cisco. In this chapter switches with high reliability of packet delivery will be discussed. Some of the networks to be presented in this chapter also support ATM. Most ATM-switches implemented until now have been designed for a different traffic pattern and network diameter than the systems we want to discuss.

6.0.1 Some common characteristics.

All the networks to be presented in this chapter put a higher pressure on high bandwidth and low latency than today's standard LANs. They all use star switches to attain this goal. Flow control is used to prevent packet loss due to buffer overflow. Circuit technology, channel technology, channel coding and allowed distances are chosen to obtain a very low error rate. The low error rate and the prevention of packet dismissal reduce the need to collect entire packets for error checking and error correction before packet forwarding. Hence all networks forward packet parts according to the "worm-hole" routing strategy 5.8 of this thesis).

Four of the networks have developed interface cards for connection to host computers. Communication with host computers takes place through standard interface buses produced by the computer industry. These network interface buses have a lower bandwidth than the network links. Thus the buses between the interface cards and the hosts make significant bottlenecks in the system bandwidth. Another very significant bottleneck is the latency due to network protocols and transfer of data between interface cards and host memories. Due to these very significant contributions to reduced bandwidth and increased latency, the need for new interface solutions are emphasised by the designers of all of the networks we will present in the following. Since most networks also are meant to serve a broad range of hosts, the interface cards must have a general nature.

For all network systems the intention has been to end up with a few customised integrated circuits. This choice has been made to utilise the reduced space, weight and power consumption (heat), and the potential for increased speed with integrated technology.

6.0.2 Comparison of key parameters from table.

The main parameters of the networks are given in table 6.1.

Four of the networks have been, or are being, built by universities while one has been designed by computer industry (Autonet). One of the university developed systems has later been commercialised (Myrinet). For all systems the intention is to implement the network components as integrated circuits in the final product. SWIPP has bypassed the PCB level and gone directly to

[1] A consortium of industry and research institutes interested in the development of the ATM standard.

ASIC design while three other systems are temporarily implemented as PCB prototypes with FPGAs and standard components (Nectar, Autonet and Telegraphos I). Telegraphos has continued to a second phase where the switch has been implemented as a standard cell ASIC. Naturally the switches and the interface cards based on standard componentsare larger than the switches and interface cards made from custom integrated circuits (Myrinet and SWIPP). Not surprisingly we see that the standard component networks consume more power. In these examples the potential for high clock rate of integrated circuits has been utilised to give a higher total bandwidth.

Table 6.1 shows some key parameters for five of the networks. The table has four horizontal sections: general, about the switch, about the interface card, and about the host. Table entries are interpreted values taken from papers[2]. Some values are exact, others are more rough (like "the size of a board") while some are missing.

Most rows should be self-explaining. In the following some of the less obvious ones are explained. *System* under the general section indicates the target application as described above. In some cases processor buffer and packet buffer are shared. This applies both to the switch and the interface card. The *buffer structure* chosen by all networks except Autonet and Telegraphos is input buffering. Telegraphos is buffered centrally with a buffer size of 1024 packets (8 bytes stored per packet) in Telegraphos I and 256 packets (16 bytes stored per packet) in Telegraphos II. Autonet is input buffered with bypass of the input queue. This gives a function and performance approximating the centrally buffered switch structure. *Cast* under the switch section is the connection mode. Unicast is maximally one output channel for each input channel. Broadcast is transmission from an input channel to all output channels. In multicast the input channel connects to any number of output channels specified by the incoming packet. *Transit latency* is the time from when a packet enters the switch until the first part leaves the same switch under the condition of no traffic collisions. *Switch rate* is here the time from a connection between one pair of input and output channels is established until a connection between another input channel and another output channel can be established. In SWIPP connections to different outputs are established independently giving a switching time of zero[3]. In Myrinet the switching time is not given, but since it is not independent it has to be above zero. Long switching time reduces the utilisation of the bandwidth, especially for short packets. *Total bandwidth* is the maximal sum of all incoming and all outgoing data. The maximal sum of different data passing through is half of this size. Three of the networks have a source routing *address format*. With source routing the address format is a description of the entire path from source to destination. Thus the routing path is entirely specified by the transmitting protocol engine. Autonet has a unique (absolute) address for each of the destinations. This network uses look-up tables inside the switches to find the local output channel. Hence a switch has an address interval for each output channel. Telegraphos uses virtual channels. Each connection is given a unique number. The switches use look-up tables to find the output number from the virtual channel number. The remaining table entries should be understandable.

6.1 Nectar

In many ways Nectar ([5], [18], [100]) is the system with the closest similarities with SWIPP. Nectar was finished in 1988 and has been in use since then. For these reasons the Nectar system will be presented more thorougly than the other networks. Nectar has been developed at School

[2]References are given in connection with the discussion of the individual networks.

[3]Here we focus on simultaneous establishing of connections for different outputs. Delays due to queuing are not included.

	Nectar	Autonet	Telegraphos II	Myrinet	SWIPP
Developer:	C.M.U. Pennsylvania USA	Dec, Calefornia USA	Inst. o.C.S, Crete, Greece	Caltec etc. California, USA	University of Oslo Norway
State:	In use since 1988	In use since 1990	Under test	Commersially availeable	Under construction
System:	Hetro, LAN	LAN	Hetro, LAN	MPP, LAN, Hetro	Hetro
Switch:	(HUB)			(Mosaic)	
Clockrate (MHz):	15	12.5	Core:25,I/O:50	25/30/50	100
Size:	2 boards 1 backplane	5 boards and one backplane $45x18x30cm^2$	$72.3mm^2$	$9.3 \times 10cm^2$	$6 \times 8cm^2$ Board with ASICs
Power:	290W	160W		5V/15W	5V/3.3V
Technology:	Standard components	Std. comp	$0.7\mu m$ CMOS Std. Cell	One ASIC $1.2\mu mCMOS$	Board with ASICs
Processor:	Dumb contr.	M68000	No	integrated	No
Processor buffer:		68kB/1MB	—	2kB ROM	—
Packet buffer:	16kB	48kB	256x16B= 4kB	64kB	2kB/15kB
Buffer structure:	Input buff.	Input buff. with bypass	Cent. buff- Dedic. VC	Input buff.	Input buff.
Arbiters:		FCFS	Round Robin		Fixed
Cast:	Uni/Multi	Uni/Broad	Unicast		Uni/(Multi)
Transit Latency:	$700ns$	Min. $2\mu s$	120ns[1]	Max $550ns$	$200ns$
Switching rate:	$70ns$	$480ns$	$40ns$(?)	$> 0ns$	Independent (0ns)
No. of channels:	16 Full dupl.	12 Full dupl.	4 Full dupl.	4/8 Full dupl.	16Full dupl.
One direction channel bandwidth:	100Mbps	100Mbps	400Mbps	640Mbps (30MHz)	800Mbps
Total bandwidth:	3.2Gbps	2.4Gbps	3.2Gbps	10.24Gbps	25.6Gbps
Fiber:	TAXI AMD 100Mbps	TAXI AMD 100Mbps or coax	No fiber, Flat cabel		
Communication board: Clock rate:	(CAB) 16.5MHz				
Board size:	15x17inch	10.5x8.5inch	"one board"	"postcard-size circuit board"	
Power:	100W				
Processor:	Sparc	AMD 29116	(Yes)	Integrated	
Processor buffer:	128kB/ 512kB		(Yes)		
Packet/Date buffer:	1MB	12kB		128kB	
Com. Protocol	TCP/IP, RMP	TCP/IP		TCP/IP - API	
Address format:	Source route	11b abs adr.	256 Virt. circ.	Source route	Source route
Inter com. board latency:	$< 30\mu S$				
Host:					
Host:	Sun/Warp	DEC Firefly	DEC Alpha	Sun3/4-Sparc	
Host bus:	VME Shared memory	DEC Q-bus	Turbo channel	VME/Sbus DMA	
Host bus bandwidth:	80/28(VME) Mbps	14Mbps		444Mbps (DMA)	
Inter host latency:	$< 100\mu S$				
Packet size:	Variable		18Bytes	16B/variable	Variable

[1]: The transit latency for Telegrahos II given here depends on synchrouous clock on connected switches. Elastic buffers between different clock domains will add time to the transit latency.

Table 6.1: The table gives an overview of some of the main parameters for some networks that are discussed in this chapter.

of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA and has been in operation since November 1988. Nectar is intended to be a "network backplane" for use in a heterogeneous multicomputer system.

6.1.1 Long term aim

The Nectar project targets the problem of heterogeneous, coarse-grained parallelism on several fronts, from the underlying hardware, through the communication protocols and node operating system support, to the communication interface of the application. The solution embodied in the Nectar architecture is a two-level structure, with fine-grained parallelism within tasks at individual nodes, and coarse-grained parallelism among tasks on different nodes. The Nectar system has been designed to obtain three goals: heterogeneity, scalability and low-latency, high bandwidth communication. The system is intended to support different hosts running a variation of programming languages, operating systems and data representations. The scalability goal implies that it should be possible to add new processing devices and network segments without removing or disturbing elements already connected. The final bandwidth goal is not specified, but new higher bandwidths should easily be achieved by adding more lines in parallel. The low-latency goal is defined to be less than $1\mu s$ through a switch, less than $30\mu s$ between communication boards and less than $100\mu s$ between hosts.

In the following we discuss the present version of Nectar.

6.1.2 The present Nectar version.

The Nectar system is based on the same two network elements as in SWIPP. In Nectar the switch is entitled "HUB" while the protocol engine is entitled "CAB" for Communication Accelerator Board.

The HUB switch compared to the SWIPP switch.

The HUB and SWIPP switches have many similarities. They are both based on a 16 channel crossbar, support both packet switching and circuit switching, use source-routing, and have low-level flow control. A difference is that the HUB has a central controller interpreting a small number of instructions, while the SWIPP switch has distributed control logic interpreting only two or three instructions. The distributed logic in SWIPP allows several connections to be established simultaneously (when there are no output channel conflicts). In Nectar the central controller can establish only one connection at a time. Simultaneous establishing of connections as in SWIPP is especially an advantage when packets are small. Another difference is that in this version of Nectar the channel bandwidth is only 100 Mbps compared to the 800Mbps of SWIPP. (Later versions of Nectar will have higher bandwidth). The time to propagate the first byte through a switch to an idle output is 700ns for the HUB and 200ns for the SWIPP switch.

The interface card: CAB

Figure 6.1 shows a block diagram for the Nectar Communication Accelerator Board, CAB. The heart of the CAB is a general purpose RISC CPU. In the current version of Nectar the CPU is a SPARC from Sun Microsystems Inc. running on 16.5 MHz. Similar to SWIPP, the CAB may be

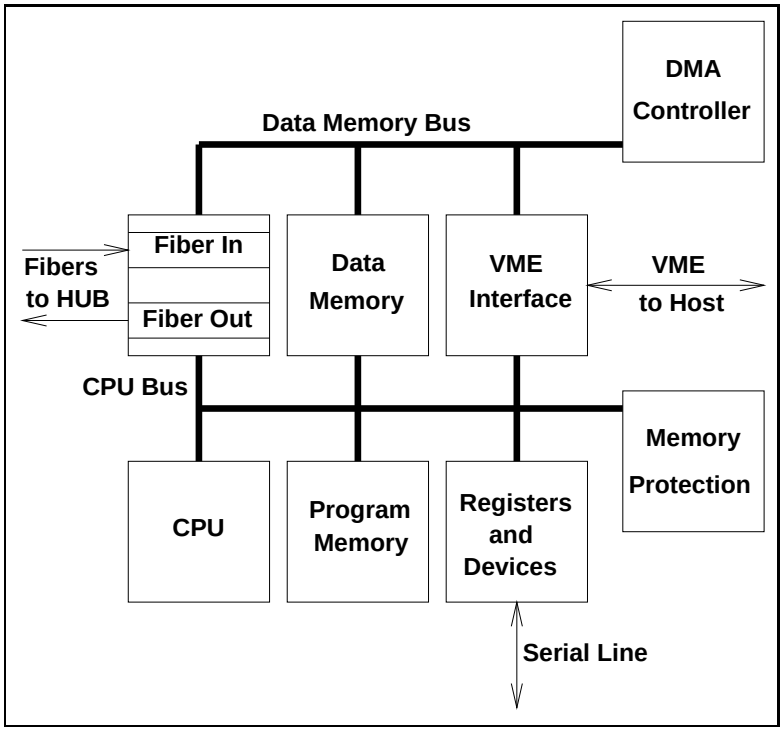


Figure 6.1: Redraw of CAB block diagram.

connected to the remaining part of the network through two fibres, one for each direction. The CAB has one FIFO for each of the directions. Cyclic Redundancy Checksums for incoming and outgoing data are computed by hardware.

Communication with the host takes place through a VME interface. A DMA controller manages simultaneous data transfer between the incoming and outgoing fibres and CAB memory, or between CAB memory and VME interface leaving the CAB CPU free for protocol and application processing. The CAB memory is split into two regions: one intended for use as program memory (128KB ROM and 512KB RAM), the other as data memory (1MB RAM). The DMA has only access to the data memory, while the CPU has access to both. One programmable memory protection is used for each 1KB page.

6.1.3 Nectar software architecture.

The CAB runtime system performs concurrent execution of activities like network interruption, transport protocol processing, and application specific computation.

Figure 6.2 illustrates the Nectar software. The *threads* package are used to support multiprogramming. It provides "forking" and "joining" of threads and mutual exclusion using locks and synchronisation by means of condition ("signal" and "wait") variables. The *mailbox* and *sync* modules are used for buffering and synchronisation. Network-wide addressing of mailboxes enables host processes or CAB threads to send messages to remote mailboxes via transport protocols. The CAB *Transport Protocols* are based on the thread tools. The *Nectarine* interface provides a consistent interface for applications on both the CAB and the host. The *CAB Device Driver* in the host operating system enables host processes to access CAB memory into their address spaces. The CAB provides a single physical address space shared by multiple threads.

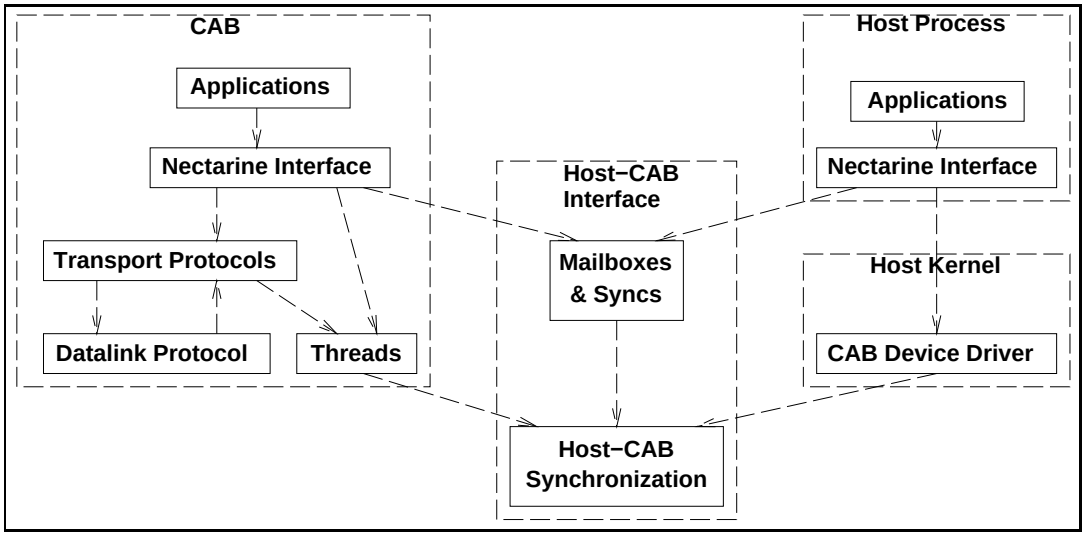


Figure 6.2: Nectar software architecture. Redrawn from [5].

The page protection system can be used to provide firewalls around application tasks.

6.1.4 Use and performance of the Nectar CAB

For communication the Nectar CAB may be used in three different ways:
a) Network Device,
b) Protocol Engine, and
c) Application-level Communication Engine.

a) Nectar CAB as Network Device

Used in this way the CAB is a traditional network device. The host driver process and the CAB server thread process share a pool of buffers. A packet is first placed in an output buffer. Then the CAB server is notified. In the other direction the CAB places a received packet in an input buffer. When the entire packet has been received the host driver process is signalled. The advantage of using a CAB as a Network Device in this way is the binary compatibility for the host: most network services are immediately available. The main disadvantage is the low efficient bandwidth compared to alternatives like the Protocol Engine. For Nectar the efficient bandwidth has been measured to 6.4Mbps ([5] 6.3). This is less than the 7.2 Mbps offered by Ethernet for the same hosts. Some of the bandwidth limitation are caused by the maximum VME bandwidth of approximately 30Mbps.

b) Nectar CAB as Protocol Engine

When the Nectar CAB is used as a Protocol Engine the Transport Protocols are off-loaded from the host and executed by the CAB. The host needs direct access to the CAB memory. Therefore the virtual pages of the host have to be translated into physical addresses in the CAB memory. One of the disadvantages of this solution is the more advanced host driver required.

An advantage is the increased bandwidth compared to the Network Device solution. In Nectar the TCP/IP protocols have been measured to have a maximum bandwidth of 24Mbps while the own protocol of Nectar, RMP (Reliable-Message-Protocol) ([5] 6.2), approximated the VME maximum of 30Mbps.

Both the Protocol Engine and the Network Device have in this implementation of Nectar been limited by the VME-bus interface. Between CABs, through the VME bottlenecks, the RMP and a version of TCP/IP without checksum, both approach the network limit of 100Mbps with increasing packet lengths. The ordinary TCP/IP protocol contains a packet checksum stored in the packet header. Until now this has required at least a double pass of the entire packet, first for generation of check sum, then during transmission of the packet contents. In Nectar the CAB-CAB transmission rate is approximately 40Mbps with the ordinary TCP/IP protocols for long packet lengths.

c) Nectar CAB as Application-level Communication Engine

In the Nectar project the CAB has also performed more application-specific code. The CABs have been used to divide labour, gather results and exchange messages between processes on the same host.

6.1.5 Protocol and network interface latency

While network bandwidth may be increased by adding more lines in parallel, it is more difficult to reduce the protocol and network interface latency. With the bandwidths and distances discussed in this thesis, latency due to protocols and network interface is in most cases dominating and always significant. Low network interface latency is important both to reduce the total latency between source and destination hosts and because the blocking part of the network interface latency reduces the efficient bandwidth. The blocking part of the network interface latency is the time used to prepare or finish the utilisation of a resource (like a channel or an interface card) where the resource can not be utilised by other requesters. This subject was discussed in connection with equation 4.2.

Several papers analysing the protocol and interface latency in Nectar have come from the members of the Nectar project. In [100] the implemented CAB with a general processor is compared to what is described as a "customised, dumb controller" for four different protocols: The Nectar-native datagram protocol, a request-response (RR) protocol, the Nectar RMP and the internet UDP/IP.

	Dgram	RR	RMP	UDP
Sparc CAB 16.5 MHz	$172\mu s$	$214\mu s$	$205\mu s$	$304\mu s$
"Dumb CAB" 16.5 MHz	$95\mu s$	$118\mu s$	$131\mu s$	$144\mu s$
Speedup	45%	45%	35%	53%

The table gives an idea of what speedup may be expected by replacing a general processor with a hardwired protocol engine. Naturally, with such a replacement a lot of flexibility would be lost, which may not be found acceptable. To put this number in relation to other figures the typical protocol latency for two workstations communicating over general networks is approximately $1ms$.

6.2 Autonet

Autonet ([89]) has been developed at the System Research Center (SRC) of Digital Equipment Corporation in Palo Alto, California, where it has been in use since 1989. Autonet is operating as a LAN with more than 30 switches connecting more than 100 Firefly workstations. The Autonet system consists of switch boards, interface boards and software for switches, interfaces and hosts. Communication takes place through ordinary TCP/IP protocols. The DEC Firefly workstations are connected to the Autonet through Q-bus interfaces.

The switch circuit.

The Autonet switch has a crossbar matrix and a general processor (a Motorola 68000). The processor takes care of address interpretation and establishes new connections. When the processor has instructed the crossbar to establish a connection (after 480ns), the processor is free for other tasks. There are buffers at each input channel. Packets may be read out from the buffers at several positions. Hence the processor may initiate bypassing of blocked packets giving a switch performance more like a centrally buffered switch than an input buffered switch. The processor lets incoming packets be forwarded to each output channel as they arrive (i.e. First-Come-First-Served (FCFS)).

The available capacity of the switch processor is also used to maintain a distributed routing algorithm. Switches periodically inform each other about their state. Each switch uses this information to generate a topology map of the network. The switches agree upon a distributed routing pattern ensuring deadlock-free packet transmission. The routing algorithm is based on a tree-like interpretation of the topology where packets are first routed 'up' and then 'down' (Described in chapter *14, Error handling,* of this thesis). There may be several routing paths between nodes.

Network interface

The main components of the network interface are the receiving and transmitting FIFO, an AMD 29116 processor, and a Q-bus interface to the host. They are connected by a common 16-bit internal bus. The interface card also consists of circuitry for error detection codes (CRC) and for encryption and decryption of data.

Host software

The host software includes a driver for the controller, the LocalNet generic LAN and the Autonet-to-Ethernet bridging software. Application communication takes place through ordinary TCP/IP protocols. In the test system, all hosts have also been connected with Ethernet network system. A software switch is used to choose between transmission over Ethernet and through the Autonet. This takes place without influencing on the host application software (except for speed-up etc.).

Next generation of Autonet

The next generation of Autonet, AN2, is going to support more common standards. AN2 can work as an ATM network forwarding ATM packets. It will also offer full-speed 155 Mbps SONET

service.

6.2.1 Discussion of Autonet.

The Autonet network consists of general processors both in switches and on network interfaces. In the switch the general processor is used to maintain deadlock-free routing tables. The switch processor may probably also be used to alter routing paths to avoid hot-spots. All of the other networks leave maintenance of routing algorithms to the network interfaces. Thus, a question is whether, and if so, how much, switch-performed routing maintenance will increase system performance. Maybe this is equally well maintained by the processors of the network interfaces. With the small distances, it is this author's belief that, the difference between when a network interface and a switch experience network congestion can be made small. Another solution, as chosen by SWIPP, is to leave such tasks to the network interfaces to make the switches small, simple and fast. In [10] another switch with a powerful processor is discussed. One conclusion made ([10], page 31, point 2) was that it was difficult to find suitable tasks for a general switch processor which could defend the cost and power of a more general processor.

One reason for using processors in switches is to discover traffic congestion early to start rerouting through other paths. However, this will result in that packets may arrive out of order, which will complicate the protocols of the receiver. Hence it is often better that the transmitter and/or receiver discover high-load spots themselves and choose between delayed transmission in order or faster transmission through alternative paths with a chance for out-of-order reception.

The communication board of Autonet has a rather similar architecture to the other communication boards. One difference is that the communication board of Autonet has an additional channel to another switch in reserve if the default connection should break. This solution makes it possible to replace switches without halting network traffic.

6.3 Telegraphos

Telegraphos (Telgraphos I: [55], Telegraphos II: [56], Pipeline buffer: [60]) is a computer communication system under design at the Institute of Computer Science, Crete, Greece. The first version, Telegraphos I, built on circuit boards with standard components was finished in 1994. A standard cell ASIC version, Telegraphos II, was submitted for processing in June 1995. The project leader had also done a preliminary design for the Autonet switch in 1987. In the following both Telegraphos I and Telegraphos II will be described. Telegraphos II, since this is the last version and Telegraphos I, since the paper about Telagraphos I ([55]) has some information not given in the other papers.

Both the Telegraphos switch and the Telegraphos network interface are significantly different from the other architectures presented.

Telegraphos consists of a switch, an interface board and software for interface circuit and host. The interface boards are connected to DEC Alpha workstations through TurboChannel I/O buses.

6.3.1 The switch circuit

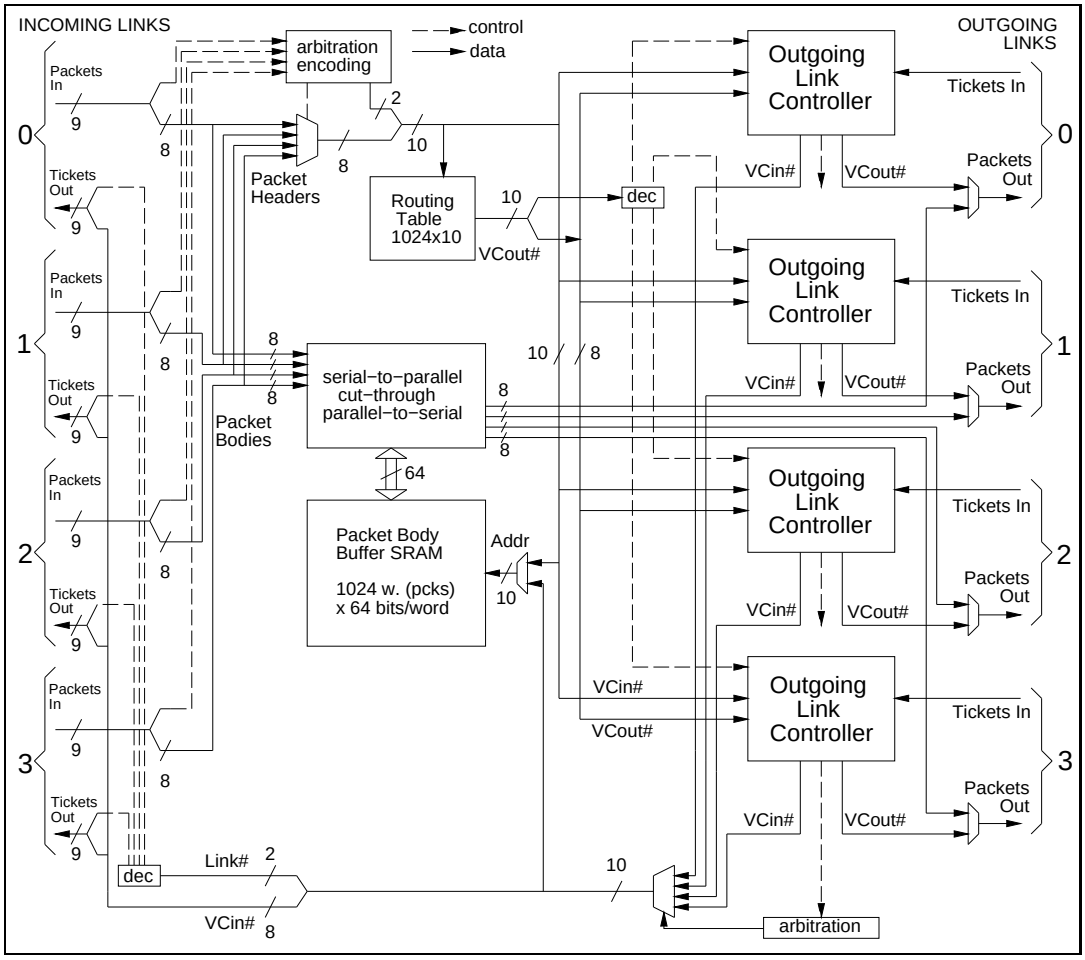


Figure 6.3: The architecture of the 4 full duplex link Telegraphos I switch. Redrawn from [55].

If we look at switch architectures disregarding the general processor element of some switches, the Telegraphos switch architecture is more advanced than the other switch architectures described in this chapter. In short the Telegraphos switch has a buffer reservation and flow control system based on virtual channels. It does not contain a processor. It has four full duplex links. The FPGA version has a link bandwidth of 107 Mbps in each direction, the standard cell ASIC has a link bandwidth of 400 Mbps while higher values are expected for future full custom integrated versions.

In the other networks, when it comes to the link between two net nodes, buffering and exchange of flow control signals is performed for the data stream as a whole. In the Telegraphos switch the data stream through each link is divided into 256 "virtual channels" treated individually. A virtual channel is a private connection from a source network interface to a destination network interface. Different segments of the source-destination virtual channel typically have different local numbers. Inside a switch, each of the virtual channels has a private buffer with a capacity of one packet. Between net nodes flow control signals are exchanged for each individual virtual channel. With 256 virtual channels for each of the four links the total switch buffer capacity is 1024 packets in the Telegraphos I version . In this version of Telegraphos each packet has a fixed length of 9 bytes. One of the 9 bytes is the virtual channel number. Since the buffer address is

the virtual channel number, this byte is redundant and does not need to be stored. Hence the buffer capacity is 1024 eight- byte packets. In Telegraphos II the packet length is increased to 18 bytes and the buffer capacity is 256 packets. If the same structure is used for Telegraphos II as for Telegraphos I, 16 bytes of each packet is stored in memory and each channel has 64 virtual channels.

The incoming link number and virtual channel number have to be replaced by the outgoing channel number and a new virtual channel number. For this use the Telegraphos I switch has a 1024 times 10 bit table with the incoming numbers used as table index and the outgoing as table contents.

As stated above, a Telegraphos I switch has one dedicated packet buffer for each of 1024 incoming virtual channels. When a packet forwarding is started from a buffer, a "ready ticket" is immediately transmitted upstream to start transmission of the following packet on the virtual channel. In this way the beginning of the next packet may enter the switch before the end of the present packet has been transmitted. Thus a large transmission from one source containing many packets can use the total bandwidth between source and destination. As discussed in 5.5.1 and by the end of the "Discussion of the Telegraphos switch", a condition for a full bandwidth utilisation is a node distance below a certain maximum.

Discussion of the Telegraphos switch.

In other switches than Telegraphos where a buffer is shared between all connections using an input channel, all connections using an output channel, or all connections in the switch as a whole, there is a possibility that one or a few connections may block all buffer space. When packet loss is not acceptable, the only way to avoid this is to have a "window"-protocol at the transmitting end allowing a maximal number of transmitted packets to be unacknowledged. This was discussed in more detail in 5.6. In the Telegraphos system each buffer is private to a source-destination connection. Hence a blocked connection will neither take buffers nor bandwidth from other connections. A blocked connection will soon be discovered at the source because of the private flow control signal belonging to the connection. Thus the source will stop further feeding. The flow control results in that the network has not been fed with more packets than needed to keep the channels busy when transmission continues. The probability of hot-spots blocking larger part of the network is reduced.

The cost for this reduced blocking in switches like the Telegraphos switch, is a more complex switch architecture and a larger minimum requirement on buffer size (given by equation 5.2). At low network load (or with a *single* heavily loaded connection) only a small part of the buffer will be in use. For such load levels a simple switch (like an input buffered switch) with a smaller buffer would give a similar performance. To avoid suffering from HOL-saturation (or bandwidth starvation) at high load, both virtual channel switches (like Telegraphos) and input buffered switches like SWIPP, require a minimum buffer space for each of the connections on a link. The difference is the size of the buffer required. For virtual channel switches the buffer size required is B_{Min} from equation 5.2 (i.e. the bandwidth times the local flow control circle time). Input buffered switches with a single lane, as the present version of SWIPP, depend on global flow control between source and destination to avoid saturation. For these switches the required buffer space per connection is $L_{global-fc-data-amount}$ from 5.6 (i.e. the bandwidth times the global flow control circle time). We have that $L_{global-fc-data-amount} = B_{Min} \cdot n$ where n is the average number of switch jumps. If the number of (requested) connections through a link is m and the Telegraphos switch has 256 virtual channels, obviously the Telegraphos switch saturates when

m is larger than 256. The required buffer space for the Telegraphos link is fixed at $256B_{Min}$. The required buffer space for the input buffered switch to avoid saturation is approximately $n \cdot m \cdot B_{Min}$. Thus, to use less buffer space than the Telegraphos switch at full blockage, $n \cdot m$ has to be less than 256.

The 256 virtual channels for each Telegraphos I link may be used for maximum 256 different source- destination computer pairs or a smaller number of computer pairs may have several virtual channels each. The LASAR network interface to be presented later (6.6 in this chapter) has 128 virtual channels. Hence 256 virtual channels for each link in Telegraphos is not much. To reduce the number of virtual channels and buffers in Telegraphos each Telegraphos virtual channel could be used for a group of network interface virtual channels. Generally, absolute upper limits such as the number of virtual channels should be avoided.

With a packet size of 9 bytes in Telegraphos I, the maximum bandwidth-distance product for Telegraphos I is $6.96 \cdot 10^9 (m \cdot \text{bit})/s$. With a bandwidth of 107 Mbps this gives an absolute maximum distance between switches of 65 m. For Telegraphos II the maximum bandwidth - distance product is $13.9 \cdot 10^9 (m \cdot \text{bit})/s$. With a bandwidth of 400 Mbps the maximum distance between switches is 34 m. In Telegraphos I the value of B_{Min} (equation 5.2) is 9 bytes and the value B_{Min} for Telegraphos II is 18 bytes. For the calculations of these distances, we presume that each connection shall have the total bandwidth when the other connections are idle. The distances are theoretical maximum values assuming no time for handeling flow control inside the switches. For such small packets, propagation through switch circuitry will take a significant part of the available time. Thus the practical maximum distance will be significantly smaller.

6.3.2 Network interface

The basic function in Telegraphos communication is "remote write" (Tele-Graphos in Greek). This function is basic both for the support of ordinary message communication and for the support of shared memory. Remote write means that arriving packets are stored directly into host memory. Thus, there are only external buffers for temporary storage until the memory bus/network can be accessed. There is no external memory for packet assembly, error check etc.

The advantages of network interfaces without additional processors and memory (like the Telegraphos network interface) are reduced cost and reduced latency due to that packet contents are copied fewer times. An important disadvantage in most implementations made until now, is that as communication increases, a larger part of the host processor time has to be used for protocol tasks. In Telegraphos some protocols are not needed due to the way in which packets are routed through the network or placed in memory. Other protocols are executed by simple Finite State Machines (FSMs). In Telegraphos no communication takes place without buffer reservation in advance. All packets "know" their memory address so that message assembly will take place automatically in host memory as packets arrive.

Shared memory support.

To support shared memory, i.e. that all computers share the same address space, Telegraphos has a tool-kit built on a set of basic commands:
• *Remote write* writes to a remote memory page. The local processor will continue without waiting for acknowledgement from the remote network interface.
• *Remote read* reads data from a remote page. The local processor will not continue before the

data have arrived. The reading may be local if a local copy has been made.
• *Remote Copy (/Prefetch/Non-Blocking Page Read)* makes a local copy of a remote page. This may be done if a remote page is often read but seldom written to. The command is not itself a read command but prepares for following read commands. Hence the local processor continues after the copy has been initiated and is first stopped by a later *Remote Read* if the page copy has not arrived.
• *Remote atomic synchronisation* commands (fetch-and-store, fetch-and-add and compare-and-swap) are used to synchronise processors with memory pages. The local processor will not continue before the complete command has terminated.

Some monitors are used to decide when a remote page should be copied for local reading, and to gather statistics in general. Each network interface has two counters for all pages, local and remote. One counter decrements on read instructions while the other decrements on write instructions. Thus, high local read activity combined with low global write activity indicates that a local copy should be made.

When a remote computer has generated a remote write to a local page, a multicast message is generated to update all copies ("Eager updating"). This is an efficient solution for pages often read but seldom written to.

6.3.3 Present version and later versions of the Telegraphos interface.

The network interfaces are connected to DEC Alpha workstations through Turbo channels. Each network interface occupies one board which includes 16MBytes of DRAM, a few SRAMs, dual asynchronous ported FIFOs, and in the order of 5 000 gate equivalents. The Mach operating system is adapted to run on Telegraphos. The second version of the Telegraphos interface will be almost equal to the first version but implemented as integrated circuits.

The network interfaces of versions I and II are separated from the internal data bus by an ordinary I/O device. Later versions of the Telegraphos system will have direct access to the internal databus. This has a number of advantages: Reading and writing of data can be initiated by ordinary memory instructions instead of the more complicated I/O instructions. This may reduce the bottleneck problem represented by the I/O devices. When processors write to memory pages which are copies, this can be discovered and updating of original and copies can be initiated in direct memory access mode.

The intention is that the Telegraphos network system can be used as a high-speed multicomputer network, as a parallel computing system and as a local area network.

Discussion of the Telegraphos Network Interface.

When used as an ordinary LAN for communication between PCs or different workstations direct memory placement of packets requires a change of system commands and system buffers. How much the potential of the Telegraphos system is utilised depends on how much work is invested in modifying and optimising system and application software. Increase in performance requiring only modification of system software is acceptable since many applications, unmodified, can benefit from the change. If applications have to be changed the cost is considerably higher. Thus the increased performance from altering application code is hard to achieve. Without a detail knowledge it may look like the Telegraphos system requires less hardware than the Nectar system, but more changes of software. If so the Telegraphos system is less portable between

different hardware configurations than the Nectar system.

The solution sketched for shared memory requires a closer connection to the computer hardware than required for message passing. The later versions of Telegraphos are planned to be connected directly to the memory bus. Direct memory bus connection requires detailed knowledge about the computer structure. Hence the number of different computer structures should be low. Different execution codes for different computer architectures would also complicate shared memory management. Hence both the shared memory concept and the Telegraphos architecture implies a system of homogenous computers.

6.4 The Mosaic switch and the ATOMIC network.

A research group at California Institute of Technology developed the *Mosaic* switch circuit. The Mosaic circuit is a switch circuit with a small powerful local processor. The purpose of the ATOMIC (ATM Over MosaIC) project was to use the Mosaic switches for Local Area Networking between a number of Sun-3 and Sun-4 workstations through their VME-bus. Some of the designers of Mosaic and ATOMIC started the Myricom company. Myricom sells the Myrinet network, which is an improved version of the ATOMIC network. We will in the following present Mosaic, ATOMIC and Myrinet.

6.4.1 The Mosaic switch circuit

The Mosaic circuit is a synthesis of a switch and a multiprocessor element. The switching function consists of 4 bi-directional channels and an asynchronous router. The processing element is a processor with a performance of 11 MIPS. The switching function and the processing part can operate independently. Packets can be routed through the switch without involving the processor, or the packets can be checked, altered etc. by the local processor. The integrated processor may also operate as a general processor taking part in a larger distributed processing task.

The Mosaic switch	
Elements:	4 full duplex channels 64 KByte dynamic RAM 2 KByte of ROM for self-test an 11 Mips processor packet interface, and a two-dimensional, self-timed router.
Channel capacity in each direction:	60 MBytes (480 Mbit)/second
Technology:	$1.2\mu m$ CMOS
Number of transistors:	1.2 million
Size:	$9.25mm \times 10.00mm$ ASIC
Power:	5 Volt
Power consumption:	0.5 W
Clock speed:	30 MHz

The Mosaic switches are packaged by tape automated bonding (TAB) in 8×8 arrays on circuit boards that can, in turn, be composed in two dimensions to construct arbitrarily large arrays of nodes. A packet may be routed from any of the four bi-directional channels or the local processor,

to any of the five sources. The address format contains a sequence of $(\Delta X, \Delta Y)$ pairs. The sign of these two numbers give the forwarding direction. The values give the number of jumps along the X and Y-axis. First the packet is forwarded in the X direction until ΔX has reached zero. Then it is forwarded in the Y direction. When also ΔY has reached zero, the packet is inspected by the local processor. The packet may be used by the local processor or the local processor may remove the empty $(\Delta X, \Delta Y)$ pair and retransmit the packet according to another $(\Delta X, \Delta Y)$ pair.

The Mosaic packets (or more correctly "cells" at this size) are 16 bytes. As in most other high speed networks for short distances the packets are worm-hole routed. Communication with neighbour nodes takes place according to a request/acknowledge protocol where each byte is acknowledged before a new is transmitted. The papers on Mosaic do not mention a possibility for a packet to pass a blocked packet arriving on the same channel. Thus the switch is classified as input buffered.

The address format is source routing. This means that the path of a packet is decided by the source node. This secures that when message parts are routed along the same path all parts arrive in correct order.

6.4.2 Discussion of the Mosaic switch.

The Mosaic switch is basically a switch processor for a dense parallel multiprocessor system. For larger distances it will suffer from reduced bandwidth due to the request/acknowledgement signalling between switches. Acknowledgement of every byte at a bandwidth of 640Mbps implies a theoretical maximal distance of 1.25m. When the signal delays due to electrical circuitry are included, the maximal distance is significantly shorter. The number of channels is low. This concentrates the traffic and makes it necessary to pass more switches on average. This increases the probability of network contention.

A dense cluster of Mosaic switches may together be regarded as one switch in a larger network. This may increase the number of channels and the capacity of each channel. The cluster will require additional link circuitry to avoid the request/acknowledgement distance-depending bandwidth. The switch will contain a number of local connections which may be bottlenecks.

Programming tools for Mosaic

The Mosaic programming tools consist of a compiler for a programming notation named $C+-$, and a distributed runtime system. $C+-$ is an extension of $C++$ that supports concurrent processes in much the same way as $C++$ supports program objects. A programming language $C+-$ and a compiler have been developed for the Mosaic circuit.

The next version of Mosaic

The next version of the Mosaic switch, named *Mosaic T* is now being designed. It is targeted for an $0.8\mu m$ CMOS technology. With 3.3 Volt power supply operating on 50MHz the power dissipation will be 0.5W.

Multiprocessors using the Mosaic circuit.

The Mosaic circuit has been used in the Intel Delta and Paragon multicomputers, the Stanford DASH multiprocessor and the MIT Alewife multiprocessor.

6.4.3 The ATOMIC local network of Mosaic switches

ATOMIC is a local area network built from Mosaic switches. It has been operating at University of Southern California/Information Sciences Institute(USC/ISI) since October 1991. The designers describe ATOMIC as a new type of Local Area Network (LAN) based on the technology used for packet communication and switching within massively parallel processors (MPPs). Myrinet is more an MPP message-passing network that can span campus dimensions than a metropolitian-area network (MAN) operating in close quarters.

To use the Mosaic switches as a LAN interface, software and hardware have been developed for connection to Sun workstations. The hardware consists of an interface board connected to the Sbus of the Sun workstation. The software consists of code for the Mosaic switches, interface code for packing and unpacking and driver code for the workstation. A mapping process is also needed to generate a map of the hosts, switches and links. This process must also generate deadlock-free routing tables for each source host for transmission to all possible destinations.

In the paper describing the ATOMIC network the clock frequency of the Mosaic switches is 25 MHz. This gives a channel capacity of 400 Mbps in each direction. Except for the reduced clock speed the Mosaic circuit should be equal to the circuit described before.

Interface board

The interface board has two full duplex channels for connection to the remaining part of the Mosaic network. These duplex channels can be used for connection to the same or to two different switches. The duplex channels can also be used to connect the interface board in a ring topology in a network without central switches. The interface board contains a memoryless version of the Mosaic circuits. These Mosaic circuits are connected to an external SRAM. The connected Sun-3 or Sun-4 workstation also have direct connection to the same SRAM through a VME bus. The interface software offers full TCP/IP compatibility. The IP address is mapped into a source address used through the network. The packets may have variable lengths. Maximum lengths can be executed by the interface board.

The Address Consultant — A common map generating process.

In the ATOMIC system one process running on one of the hosts, the Address Consultant (AC), is responsible for mapping of channels and switches. It generates and distributes address tables for all interface boards. Local Area Networks are dynamic since workstations are inserted and removed. Thus changes in the network must be discovered and correct routing tables generated and distributed.

Regular scaleable two-dimensional topology.

The ATOMIC designers recommend that the switches are connected in a two-dimensional topology. The two-dimensional topology is easy to scale to meet increasing demands on capacity. It is also adapted to deadlock-free routing algorithms where different dimensions can be given fixed directions. Two-dimensional topologies are recommended in front of higher multi-dimensional networks due to the high pin numbers and difficulty in connection of higher dimensional networks.

Low error rate

ATOMIC has an impressingly low error rate. During test more than $1.0 \cdot 10^{15}$ bits was transmitted without errors. This represent 29 days of constant transmission on a 400 Mbps channel.

The experience from ATOMIC.

Five major conclusions about ATOMIC were made for the design of a new network ([10] p. 31):
• Asynchronous communication based on request-acknowledgement signalling between network elements could not be used. The network bandwidth was decided from the propagation time back and forth, hence the element distance was limited.
• The ATOMIC topology with one-dimensional chains of interfaces and two dimensions of multicomputer switches was too complex for real network mapping. It was also difficult to insert and remove switches without interfering seriously with the system.
• No proper use could be found to defend the substantial computing power and memory in the Atomic switches.
• It was clear that the network interface required a DMA engine. It was also concluded that to managing the network interface it was crucial to have the possibility of performing complex control programs.
• The TCP/IP protocols seriously limited the end-to-end data rates that the network could achieve.

6.5 Myrinet

Some of the researchers taking part in Mosaic and the ATOMIC project started the Myricom company. Based on their experiences they designed and are now manufacturing a new network system: Myrinet [10].

6.5.1 The Myrinet switch and packet routing.

Two full custom circuits are the basis for the Myrinet switches: A crossbar circuit performing the switching function and a dual-Myrinet-interface circuit. The circuits have been designed in a $0.8\mu m$ CMOS technology. Currently 4- and 8-port switches are for sale while 16- and 32-channel switches are under design.

The SWIPP switch has more similarities with the Myrinet switch than any of the other switches

that are discussed here. Although not clearly stated, it looks like the Myrinet switch has an input buffered architecture. The flow control schemes are very similar. For every data stream flow control signals are transmitted through the peer channel in the opposite direction by insertion of control characters. A difference is that Myrinet requires dedicated flow control symbols while SWIPP inserts a flow control bit in any control symbol. Both network systems generate flow control signals based on measurements of "almost-full" taps. A difference is that while SWIPP is using the same tap for "stop transmission" and "restart transmission", Myrinet has a separate tap for "restart transmission" to get a hysteresis. In SWIPP the hysteresis is decided by the flow control circle (flow control reaction time). The packet formats are also similar. Both have a packet header containing a source route where addresses are removed as the packet propagates. They both allow an arbitrarily long payload.

6.5.2 Discussion of the Myrinet switch.

In [10] the distance between the "restart transmission" and "stop transmission" taps is not given. If we expect them to be close, an 8kB buffer pr. channel with a maximum bandwidth of 640Mbps indicates a theoretical maximum distance of 618m. When the delays in the electrical circuitry and the distance between the taps are included, the maximum distance between net nodes should still be several hundred meters.

The parts of the Myrinet switch described in [10] shows, as stated above, many similarities with the SWIPP switch. Since SWIPP is discussed throughout this thesis it should not be necessary to repeat this discussion here.

6.5.3 The Myrinet network interface.

The Myrinet network interface is connected to Sun Sparc stations through Sbus interfaces. Interfaces to other buses, PCI included, are under development.

The network interface is based on a full custom circuit and a 32kx32bit fast static RAM. The full custom circuit, entitled LANai, consists of a DMA engine, a processor, a packet interface and a Myrinet interface. The circuit has an address bus and a data bus which can be reached from the host. The DMA is used for rapid transmission to and from host memory. The processor executes network protocols while the packet interface performs the most common or most time sensitive commands like packet segmentation and packet reassembly. The Myrinet interface performs the functions required for all network links in the Myrinet network like input buffering and exchange of flow control signals.

The static RAM contains packets, the Myrinet Control Program (MCP) and queues for communication. The MCP program is executed by the integrated processor on the LANai chip. The static RAM may be accessed twice in each clock period, once by the host processor and once by either the LANai processor or LANai packet interface. In [10] it is claimed that the circuit can reside as a bus slave on most 32-bit memory or I/O buses.

For communication through the network, the hosts may use either the IP (Internet Protocol) or an own Myrinet protocol designed for a Myrinet Application Programming Interface (API). The Myrinet protocol has been designed to have less latency than IP. Communication between the host API and the LANai MCP takes place through 6 queues situated in the network interface RAM. Three of the queues are only written to by LANai and read by the host while the other three are used with the opposite access.

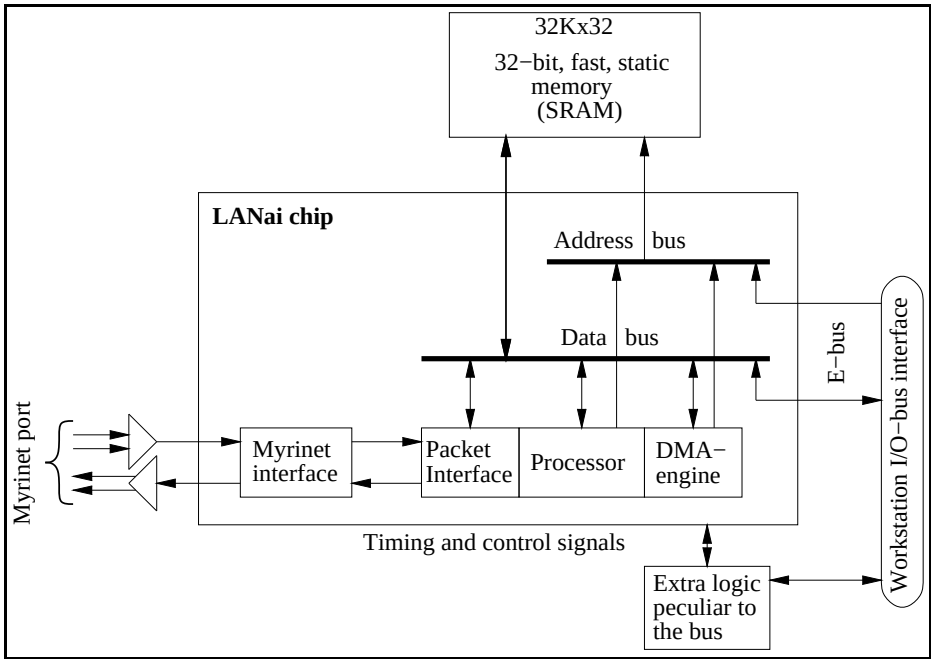


Figure 6.4: The Myrinet network interface. Redrawn from [10].

An example of tasks handled by the LANai MCP upon a transmission request from the host:
- Perform a gather operation of data from a table of host addresses and word counts.
- Generate Internet checksum and insert in correct position.
- Transmit the packet to a distant host given a host identity.
- Signal end of task, optionally generate an interrupt signal.

Network interface bandwidth and network interface latency.

The traditional TCP/IP and UDP/IP protocols have been designed for connections with lower bandwidth and higher error rate. In newer networks with higher network capacity and lower network propagation time the bandwidth limitation and latency of these protocols become significant bottlenecks. To reduce this problem the Myrinet researchers have developed the API (Application Programming Interface) network interface for lower latency and higher bandwidth.

Table 6.2 shows some measured results for some hardware and protocol configurations. The first row shows the capacity of the network links. The remaining part shows the performance for a Sbus interface board. The raw speed is the data rate between hosts when the part of the time used for packing and unpacking has been reduced significantly. This is achieved by transmitting as long packets as possible. Introduction of a DMA engine for fast transmission to and from the host memory increases the capacity from 380 Mbps to 444 Mbps. The next box shows the reduction in efficient bandwidth due to protocol latency. The efficient bandwidth is measured for a host which is constantly transmitting 8-kbyte packets. An ordinary UDP/IP version gives an efficient capacity of 55 Mbps. The TCP/IP protocol is an experimental protocol with no packet copying. The measured capacity for the TCP/IP is 70Mbps. At last, the Myrinet API attains a bandwidth of 250 Mbps.

Myrinet link data rate	640Mbps
Raw speed test	380 Mbps
Data rate with DMA	444 Mbps
UDP/IP (1-copy, hardware checksum)	55 Mbps
TCP/IP (0-copy, hardware checksum)	70 Mbps
Myrinet API	250 Mbps

Table 6.2: Myrinet network interface data rates

	$L_{pk} = \infty$	$L_{pk} = 8KB$					
	Effective Bandwidth $B_{\text{eff}} = B$	Effective Bandwidth B_{eff}	Network Interface Latency t_{lat}		Time through a point. L_{pk}/B		Switch Latency t_{sw}	
UDP/IP	380MBps	55MBps	$995\mu s$	86%	$168\mu s$	14%	$0.5\mu s$	0.04%
TCP/IP	380MBps	70MBps	$746\mu s$	82%	$168\mu s$	18%	$0.5\mu s$	0.05%
API	380MBps	250MBps	$88\mu s$	34%	$168\mu s$	65%	$0.5\mu s$	0.2%

Table 6.3: Estimated time for different parts of the transmission of 8-kbyte packets.

6.5.4 Network interface latency and efficient bandwidth.

Based on table 6.2 copied from [10] we may estimate the network interface latency for the different protocols.[4]

Table 6.3 shows estimated time for different parts of a transmission. We see that the network interface latency for the common UDP/IP is ten times as large as for the Myrinet API protocol. The interface latency shown is the non-overlapping part where no other packet transmissions/receptions take place simultaneously. The contributions of the non-overlapping protocol parts lead to a factor 5 difference of efficient bandwidth. We also see that the interface latency is between 160 and 2000 times as large as the latency through a switch.

From these numbers it is easy to conclude that for a relative reduction of one of the numbers a reduction of the interface latency will give the largest increase in performance. Switch latency is least important.

A method to increase efficient bandwidth without reducing latency is to reduce the non-overlapping part of the interface latency. This may be done through pipelining.

Table 6.4 shows the same interface and switch latencies for a stream of 53-byte packets. Here the interface latency is dominating for all protocols and decides the low efficient bandwidth. The channel potential of 380MBps is almost wasted. Clearly neither of the protocols can be used to generate 53-byte ATM packets. Thus packet segmentation and preparation has to be done by specialised hardware.

[4] The author of this thesis interprets from [10] that API, UDP/IP and TCP/IP all have been measured for 8KB packets and with no DMA between network interface and host.

	$L_{pk} = \infty$ Effective Bandwidth $B_{\text{eff}} = B$	$L_{pk} = 53B$(ATM Standard Cell)					
		Effective Bandwidth B_{eff}	Network Interface Latency t_{lat}		Time through a point. L_{pk}/B		Switch Latency t_{sw}	
UDP/IP	380MBps	0.43MBps	$995\mu s$	99.8%	$1.14\mu s$	0.1%	$0.5\mu s$	0.05%
TCP/IP	380MBps	0.58MBps	$746\mu s$	99.8%	$1.14\mu s$	0.15%	$0.5\mu s$	0.07%
API	380MBps	4.85MBps	$88\mu s$	98.2%	$1.14\mu s$	1.3%	$0.5\mu s$	0.6%

Table 6.4: Estimated time for different parts of the transmission of 53-byte packets.

6.5.5 Interface boards

Myrinet has designed interface boards for connection to the Sun Sparc stations through Sbus. It also has under development an interface card to the PCI local bus used in 486 and Pentium PCs.

6.6 LASAR-155: A commercial ATM - PCI interface.

The PM7375 LASAR-155 [35] is an example of current state (1/96) of commercial network interfaces. It is an integrated circuit for connection between the Intel PCI -bus used for the last generations of the Intel processors and the ATM 155 Mbps standard. Some additional circuitry is required to drive lines or convert between electrical signals and light.

The integrated circuit contains all protocols necessary for connection to 155 Mbps ATM. It contains a SAR-engine (ATM cell Segmentation And Reassembly), a table for Virtual Circuit parameters and a PHY (Physical layer interface) for both PCI and ATM. The circuit talks directly to a 32-bit, 33-MHz PCI bus. Data are placed directly into a part of the host memory used as buffer area. Transfer is controlled and performed by the PM7375 as a non-interfacing DMA transfer. A selected part of the host memory is, under the control of the PM7375, designated as a linked-list buffer, where scatter/gather functions are performed.

The PM7375 uses a DMA to get data from and put data directly into the host memory at full PCI speed (4.2Gbps). Cells are placed directly into host memory. No additional memory is used except for two FIFOs for temporary storage when bus or network is occupied. The PM7375 contains a 96-cell FIFO in the receiving direction and a 8-cell FIFO in the transmitting direction. This corresponds to received data during a $270\mu s$ PCI bus delay for bus request. In the transmission direction a data amount of 8 cells will occupy the ATM network for $23\mu s$.

The circuit contains clock recovery and clock synthesis units, SONET/SDH framer, overhead and cell processing functions st STS-3c/STM-1(155.52Mbps) and STS-1 (51.84Mbps) rates. Thus only simple line drivers/receivers for electrical wires or simple converters between electrical and optical signals are required.

The PM7375 contains a status table for 128 receiving and 128 transmitting virtual channels. Each virtual channel belongs to any of four high priority and four low priority queues. Hence guaranteed bandwidth may more easily be given to high priority virtual channels.

An optional 8-bit microprocessor may be connected to configurate, control and monitor the PM7375. With PM7375 in bus-slave mode, the host processor can access the PM7375 registers and communicate with the optional microprocessor via a mailbox mechanism provided by the integrated DMA controller.

6.6.1 Discussion of the LASAR-155 PM7375.

The PM7375 has similarities with the Telegraphos interface: Virtual channels, DMA placement of packets into host memory, no external memory for temporary storage and reassembly and sorting of packets directly into host memory. In an implementation where only standard system procedures are modified for Telegraphos, the functionality would be equal. A difference is that Telegraphos has the potential of letting applications request data delivered anywhere in host memory.

It is clearly a trend for present and coming network interfaces to avoid external memory and put and get packet contents directly to and from host memory. This leads to less latency. Expensive, fast memory can be avoided and memory can be utilised better by having one large instead of several smaller memories.

6.7 The Credit Net ATM Project

Also worth mentioning is the Credit Net ATM project. The project takes place at School of Computer Science at Carnegie Mellon University (CMU) and is based on the knowledge attained from the Nectar project. The network consists of switches built by Bell Northern Research. The switches have an architecture similar to the Telegraphos switch with flow control signals, "credits", transmitted between buffers reserved for specific virtual channels. The network interface, a PCI host adapter, is built in co-operation between Intel Architecture Labs and CMU. The network forwards ATM packets with a bandwidth of 622 Mbps.

6.8 ATLAS I

ATLAS I [58] is a one-chip ATM circuit under design within the ASICCOM[5] project. This project is a part of the European Union ACTS[6] program.

The ATLAS circuit has 16 input and 16 output channels. The bandwidth of each channels is 622 Mbps. It is being implemented in $0.5\mu m$ CMOS and the estimated size will be $225mm^2$. It is intended used in systems ranging from Wide Area Networks (WANs) to Local Area Networks (LANs) and Desktop Area Networks (DANs). For longer distances between switches, additional circuits (LLI = Long Link Interfaces) with buffers are required. The switch core is operating on 50 MHz (2 bytes wide buses) while the switch I/O is operating on 84 MHz (1 byte wide bus). The transit time for a bit through the circuit is $340ns$. Access to tables etc. makes it impossible to serve several inputs simultaneously. All inputs and all outputs are served during 640ns. Thus the time between establishing of new connections is 40ns.

The ATLAS I switch uses a sort of virtual channels entitled "flow groups". A flow group is a

[5] ATM Switch for Intergrated Communication, COmputation and Monitoring.
[6] Advanced Communication Technologies and Services.

set of connections sharing one or more links in their path. Preferably connections to the same destination should be grouped together. Each flow group has one flow control "credit" allowing forwarding of one ATM cell. Each of the 16 channels has 4096 flow groups. The circuit has a storage memory for 256 ATM cells of 53 bytes (total of 13 568 bytes). A separate type of flow control credits are used for the storage buffer (one cell entry is one credit). To forward a packet downstream a connection requires both a credit for the flow group and a credit for the buffer.

The cells forwarded belongs to one of three priority classes. The traffic transferred in the highest and in the medium priority classes are data from sources where congestion is expected to be rare and service guarantees are possible. The medium and low priority classes use flow control. A part of the buffer is reserved for traffic with the highest priority. This class uses no flow control, so when the buffer part is full, packets will be dismissed. Traffic from a higher priority class are always transmitted in front of traffic from a lower priority class.

The circuit has 54 ready queues for outgoing traffic. A queue contains the identification and address of a packet which can be forwarded when flow control tokens are received. The 16 external channels and one internal channel has one ready queue each for each of the three priority classes. These are unicast queues for one destination. In addition each of the priority classes has one multicast queue.

6.8.1 ATLAS I compared to SWIPP.

Both the ATLAS I switch and the SWIPP switch are intended to be implemented as one ASIC. Both switches use flow control (except for the high priority class in ATLAS I) to guarantee delivery. ATLAS I has 4096 virtual connections on each channel to allow bypassing of blocked connections. SWIPP has one shared connection in each channel. The advantage of ATLAS I is better performance at high load.

The ATLAS I chip area is estimated to about $225mm^2$ ($0.5\mu m$ CMOS) of which $75mm^2$ is used for control, $45mm^2$ is used for buffer, and $70mm^2$ is used for pads. At least 7 % of the channel bandwidth is used for flow control and handling of virtual channels. The buffer capacity of 256 ATM cells gives an average storage capacity of 16 ATM cells per channel.

A SWIPP switch core of $100mm^2$ designed for $0.5\mu m$ CMOS at 2.5V has an estimated area of $32mm^2$ for control logic, leaving $68mm^2$ to buffer. These $68mm^2$ gives 50.4 kB of buffer space (3 150 B per channel). With one 5-byte SWIPP header per ATM cell, this gives 53 ATM cells per channel. If the SWIPP switch is increased to the same size as ATLAS I and the new additional space is used for buffer, the buffer per channel will be 8 196 B per channel or 141 ATM cells.

In an ATLAS I network a heavily loaded connection can take at most one ATM buffer cell per switch. The maximum number of heavily loaded connections before the switch saturates is 256. In SWIPP, a heavily loaded connection without source-destination flow control can take all buffer space. Thus, global source-destination flow control is required in SWIPP. This global flow control limits the data amount fed into the network on a blocked connection (5.6). With large buffer space, the data amount of a blocked connection can be kept close to the contention point. Switches upstream can be released for unblocked connections. We may make an example as follows: The connection passes 15 switches (i.e. the maximum of SWIPP), the distance between switches is 25m (arbitrarily chosen) and the bandwidth is 800 Mbps. With these values, a source with global flow control will feed a blocked connection with approximately 1 000 bytes. Thus a $100mm^2$ chip can keep the data for a little more than three blocked connections (per channel). To compare with the ATLAS I switch, we may increase the SWIPP switch area to the size of

the ATLAS I switch and reduce the SWIPP effective bandwidth to the effective bandwidth of ATLAS I. For these values each SWIPP channel may store the data of more than 11 blocked connections.
In a network of ATLAS I switches, there are upper limits to number of channels and distance between switches, within which no buffer saturation occurs. These limits can not be exceeded. To keep saturation local in SWIPP, there is an upper limit to the product of the connection number and the distance from source to destination. In SWIPP, the limit may be passed, resulting in expansion of saturation to other switches.

ATLAS I is a complex switch architecture which with its qualities off-loads the global flow control from the protocols of the source and destination . It also simplifies the topology design, since the requirements for high performance is less critical than for SWIPP. To give high performance at high load, SWIPP requires more complex protocols in the protocol engines. SWIPP also requires a more careful topology design. To reduce HOL-blockage, SWIPP requires more switches than ATLAS I needs. On the other hand, for long connections ATLAS I requires additional circuits (LLI = Long Link Interface), while SWIPP can handle long connections directly. With the flow control scheme of ATLAS I the distance-bandwidth product is $41.8 \cdot 10^9 (m \cdot \text{bits})/s$. With an efficient bandwidth for ATLAS I of approximately 580 Mbps the absolute upper theoretical distance between switches is 72 m. The practical distance is significantly shorter. For a SWIPP circuit of size $100mm^2$ the distance-bandwidth product is $1.2 \cdot 10^{12} (m \cdot \text{bits})/s$. With an efficient bandwidth of 800 Mbps, the theoretical upper distance is 1 500 m.

6.9 Macro switches / Mini topologies.

Input buffered, centrally buffered and output buffered switches and combinations of these normally have one or more non-blocking switching matrixes. It is possible within this class of switches to approach the theoretical minimum waiting time decided by the traffic pattern and the bandwidth of the output channels. One of the disadvantages of these architectures is that the complexity grows with N^2, where N is the number of input / output channels. As N grows, there will be a limit where the cost in complexity is so high that other architectures have to be considered. Two of the possibilities at this point are:
• Mini topologies: Creating a topology of several of the switches ("mini switches") mentioned above.
• Macro switches: Using a large switch where the architecture grows with a complexity smaller than N^2.

The mini switches can be made non-blocking, they has the potential for the highest performance but grows in complexity with a factor of N^2. The macro switches have internal blockage and / or packet dismissal have a lower performance potential, but grow with less than N^2 in complexity. Since the non-blocking mini switches can not be put together in a non-blocking topology, such a topology will offer a less ideal performance than the individual switches alone. A macro switch may offer a better performance than a topology of mini switches.

In the following we will focus on the macro switches, examplified by a few modern and common structures.

6.9.1 The Knockout Switch.

The Knockout switch [111] contains a matrix of 2×2 switches. This matrix may be regarded as a reduced matrix of an output buffered switch. In a real output-buffered switch, the buffer of any output channel should be capable of receiving one packet from each input channel simultaneouly. This would require a large switch matrix and a large buffer when the number of channels increase. In the Knockout switch, the number of packets which can be received simultaneously, is reduced. When capacity is exceeded, packets may be buffered for later, but some packets can be lost.

The Gauss switch from the Dutch telephone company ([81] pg. 107-109) is an improvement of the Knockout switch.

6.9.2 Banyan switches.

The basic Banyan ([34][7] [110][8]) network consists of $\log_2 N$ levels of 2×2 switches with $N/2$ switches in each level. Thus the complexity in paths and switching elements grows with $(N/2)\log_2 N$. This network has only one path from any input to any output channel. Segments of this path are shared with connections between other input / output channel pairs. The performance shrinks rapidly with increasing N. The architecture can be improved by having:
- buffers in each switch,
- back-pressure flow control system,
- several networks in parallel to offer multiple paths between input and output, or
- multiple switches and links to provide multiple paths internally in the network,

6.9.3 Batcher-Banyan switches.

An alternative is to increase the network with a Batcher network. A Batcher network offers a non-blocking behaviour by preventing that multiple packets reach the same output simultaneously. In the Starlite switch ([44] [43]), a trap network between the Batcher network and the Banyan network removes all but one packet for each destination. The removed packets are forwarded to a buffer and enter the Batcher network as soon as possible. The Sunshine [32] switch has k parallel Banyan networks at the output. Hence, k packets can be routed to each output. Low-cost high-density RAM memories are used to smooth out bulk arrivals of packets on each channel output. The packets exceeding k packets to each destination are cached in a trap network and routed to a buffer. A Sunshine [32]switch with 32 input and output channels and a bit rate of 170 Mbps on each channel, is under implementation. The switch consists of nine different full custom $1.2\mu m$ CMOS ASICs. A disadvantage of the Sunshine switch is that communication is not guaranteed. Simulations in [32] shows that with random arrivals (Poisson distributions), $k = 2$, and 0.33 recycle buffers per input channel, a 10^{-6} cell loss probability is achieved.

The complexity of the Batcher module grows with $N \log_2 N^2$. Thus is grows faster than the Banyan module, but slower than a Crossbar matrix. The Batcher module has the disadvantage that packet propagation has to take place synchronised for all packets. This complicates the implementation when the number of input channels and the clock rate increases.

[7] Basic Banyan
[8] General tutorial.

6.10 Concluding comparison of SWIPP and the network system examples.

In the beginning of this chapter, in table 6.1 and in the text in connection with the table, SWIPP was compared with the other networks. Throughout the chapter, the example networks have been compared with SWIPP when found natural. In the following we will make a concluding comparison. First we will compare the switches and then make some conclusions about the Network interfaces (Protocol Engine).

6.10.1 The switches.

Compared to the other switches, SWIPP has been designed with a harder pressure on utilising technology. SWIPP has a clock rate (100MHz) more than three times the clock rate of the second fastest switch. SWIPP has also been designed as integrated circuits while most of the others are still at the prototype level, built by standard components. The more conservative technologies chosen for the other switches may be interpreted as a wish to confirm correct behaviour first. Thus a similar pressure on integration and maybe also clock rate may be expected later.

In the following we will focus on the more architecture dependent characteristics.

High bandwidth and low propagation time.

High bandwidth and low propagation time is substantial for network performance. High bandwidth is especially important since it both reduces the time to propagate a packet and reduces contention problems dramatically. The lower bandwidth of the other switches (than SWIPP) is probably due to the choice of a more conservative technology and not to lack of understanding of the importance of bandwidth. Increased bandwidth will probably be a natural choice and will be implemented on the other switches with time.

Fast establishing of new connections.

The SWIPP switch has a simple and distributed arbitration and control logic for establishing of new connections. This is different from all of the other switches where only one new connection can be established at a time. The advantage of this choice for SWIPP is most beneficial for short packets. The sequential establishing of connections for the other switches may give a significant reduction in effective bandwidth, especially for the switches which use short fixed packet lengths.

An important element in supporting distributed control and simultaneous establishing of new connections is the choice of source-routing. With source routing the packets will contain the necessary addressing and no look-up in a centralised table is required. It is this author's belief that the advantages of source-routing outrule the disadvantages, and that rerouting and readdressing may also be implemented if necessary. For longer distances, source-routing may require more address bits than other address schemes. The author does not believe this to be a problem for typical topologies and average routing patterns. This is discussed further in 7.3.3.

Flow control.

To simplify network protocols and secure transmission of data, packet dismissal has to be avoided. Thus flow control is required. All systems described in this chapter use flow control. SWIPP and Myrinet use similar flow control strategies. The disadvantage of this specific strategy is that the minimum buffer space required is more than twice the minimum sizes which can be achieved with other flow control strategies. This is a disadvantage if the available buffer space per channel is in this area. If so, this should be a temporary problem. Memory per chip, both for pure memory circuits and for mixed circuits, is increasing faster than the clock rate ($\sim$ bandwidth $\sim$ minimum buffer space). Thus, if not now, the buffer capacity in the future should be several times the minimum buffer capacity required.

An advantage of this flow control strategy is that it occupies a smaller part of the bandwidth than other flow control strategies. Another advantage is its independence of packet size.

Variable packet size and input buffering.

SWIPP uses variable packet size. Except for Nectar it may look like all the other networks use fixed packet sizes. It is not completely clear from the papers about Nectar whether it is using variable packet lengths at the physical level or whether the packet lengths are variable at a higher level (i.e. a variable message).

As explained in 5.9 of this thesis, variable packet lengths give a more efficient utilisation of bandwidth. Variable packet lengths are easily managed in strict FIFO buffers.

In common with Nectar and Myrinet, SWIPP has an input buffered switch architecture with strict FIFO buffers. With simple flow control systems this results in the HOL blocking as described in 5.4.1 of this thesis. Local bypass may give some improvement, but the saturation effect can not be completely avoided when flow control is used for shared buffers. To improve the performance significantly, we have to reserve buffers and flow control signals for dedicated connections or groups of connections. By using such flow control information, packets to available channels may bypass the blocked packets.

Telegraphos

Telegraphos is the switch most significantly different from the others and deserves an individual discussion. In Telegraphos flow control signals and buffer segments of fixed size are dedicated to individual source-destination connections. Thus connections can not block buffer space belonging to other connections. The buffer and flow control structure allows bypass. Consequently this switch is not suffering from HOL (5.4.1) saturation like the other switches. Clearly the Telegraphos switch has large advantages compared to the other switch architectures for the most difficult load distributions. At simpler load patterns the switch will have a performance closer to the performance of the other switches. The local flow control gives implicitly a global flow control between the source and destination. No additional global flow control is needed.

However, the high performance is not without disadvantages. The major disadvantage is the large minimum buffer size required. The buffer should have as many buffer segments as possible to allow a high number of connections. (Actually, having a hardware maximum on the number of connections is generally a large disadvantage.) Each buffer segment should be large to support high bandwidth, acceptably long distance between net nodes, and high utilisation of the available

bandwidth. The distance-bandwidth product for the chosen packet length (9byte) is small ($< 7128 Mbps \cdot m$). Thus, to have a bandwidth of several hundred Mbps at distances of several tenths of meters, the packet length will have to be increased. Satisfying the bandwidth-length relation is required if a heavily loaded connection shall have all the bandwidth when the other connections are idle. The short packet length and the intensive exchange of tokens reduces the efficient bandwidth. The bandwidth utilisation[9] is 80 % for Telegraphos I and 90 % for Telegraphos II. If the packet size is increased to the size of an ATM cell the bandwidth utilisation would be 96 %. For SWIPP it may become 100 %.

The high performance of Telegraphos has a high cost in large minimum requirement on memory. Only small parts of the memory are utilised for medium and low load. Since technology development allows more and more memory on a circuit, the future may be on the side of architectures like that of Telegraphos. This is the case if the memory on a circuit increases more than the product of link bandwidth and the number of simultaneously supported connections through a link.

A more probable solution[10] is a compromise between Telegraphos and the other switch architectures. The sizes of buffer segments are increased to give a higher distance-bandwidth product and a higher utilisation of the bandwidth. Each buffer segment is shared between a group of connections (virtual channels). (This is the choice for the ATLAS I switch [58]). In this way buffer blocking is limited, so that most other connections may have some buffer space. There is no upper limit to the number of connections. The disadvantage is that this allows buffer blockage inside each group of connections. Another disadvantage compared to Telegraphos is that the packet headers probably have to carry more information. Thus the conclusion is that with this strategy increased memory per chip is utilised to reduce blockage effects and increase utilisation while no connections are closed out.

Local switch processor.

The conclusion of this author (which is also stated in some of the papers) is that embedded processors may be used in switches while having a general processor in each switch can not be defended. The exception is when the switch is an element in a massive parallel processor. The routing map maintenance done in some of the switches described, can be moved to the processors in the network interfaces. When this choice is made, it is even easier to defend the source-routing address format.

6.10.2 The network interfaces

The network interfaces described in this chapter may look very different. At the same time the impression is that the conclusions, and the planned future versions, go very much in the same directions. Thus, the differences in architecture are due to different temporary goals.

[9] The bandwidth utilisation is D/(D+H+T). D is the data size of a packet, H is the header size and T is the size of the token.

[10] Also proposed by the Telegraphos designers.

The limitation of standard I/O buses.

In the network systems heading for fast prototypes, the network interfaces are connected to the main computers through standard I/O buses like VME, Qbus, Sbus and Turbo channels. These buses give a significant reduction in effective bandwidth compared to the bandwidth of the network cores. The Myrinet network has a bandwidth of 640Mbps while the bandwidth is reduced to 380Mbps due to the standard interface bus. Thus a clear conclusion is that the network interface cards should be connected directly to the host memory bus. This is not a choice without disadvantages. Direct memory connection requires that the hardware is designed more specifically to the host. Thus, some of the flexibility and generality is lost and portability to new computers is not easily obtained.

Direct memory access without multiple copying.

In some of the newer network interfaces like Telegraphos and LASAR it is emphasised that data are placed directly into and read directly from host memory. The data are not temporary stored inside the network interface (except for some FIFOs when memory bus or network are busy). In this way multiple copying is avoided and the total communication latency reduced.

Old protocols contribute significantly to latency.

Another conclusion is that for the new high performance networks the old network protocols contribute with a very significant part of the communication latency. In Myrinet, the $995\mu s$ delay, due to execution of the UDP/IP, reduced the effective bandwidth for 8kbyte packets from 380Mbps to 55Mbps. The part of the communication time used on the UDP/IP was more than 85%. Implementation of new and faster, secure network protocols is therefore essential. For the Myrinet protocol, the execution time was reduced to $88\mu s$. Newer standard protocols are under development (Not discussed in this chapter: IP version 6 [50] and RTP (Real Time Protocol) [51].). This author expects numbers similar to those for Myrinet also for the other networks.

General processor and/or specialised hardware.

The answer to the question whether the network interface should contain a general processor or not, depends on the functions given to the network interface. All network interfaces except Telegraphos and LASAR have processors, but for different reasons. For LASAR an optional processor may be connected. However, there is a clear consensus that specialised circuitry is required to speed up the execution of the most common tasks. For Nectar four different protocols expected to be used frequently, were simulated with a general processor and a "dumb" controller. The specialised controller used between 35 % and 53 % of the time of the general processor. Here it should be noted that also for the general processor, dedicated hardware took some of the low level tasks.

Buffer access point for host processor.

There are two main ideas of how the host computer is going to access packet data. Either the host computer accesses the data in its own memory, or the data are accessed in the network

interface memory. The first alternative requires small modifications of the binary code. The second requires remapping of virtual pages.

Valuable background for SWIPP

The network interfaces presented in this chapter all have their advantages and disadvantages. The experience collected from these network interfaces give a good background for the choices we have made for the SWIPP Protocol Engine.

6.11 Networks not mentioned here.

In the previous chapter and the previous part of this chapter, the author has discussed the networks believed to be most relevant to this project. The following contains some references to where more information can be found.

6.11.1 Books.

Asynchronous Transfer Mode [82], 1993, by Martin De Prycker is considered to be one of the most important books on ATM. *Gigabit Networking*[81], by Craig Partridge gives a good overview from one of the most central contributors in the field. Chapter 16 of [81] contains a list of important network research projects.

6.11.2 Publications.

In the author's opinion, the most important sources are *IEEE Network*, *IEEE Journal on selected Areas in communication*, *IEEE Transaction on Communications* and *IEEE Computer*. Several special issues of *IEEE Micro* are also interesting.

6.11.3 Internet.

Obviously, network oriented companies are present on the Internet and it should not be difficult to find their homepages by using the search engines.

Several universities maintain lists of research projects etc. The author has found three home pages of special interest:

The http page of the CPU Info Center at the University of Berkeley:
http://infopad.eecs.berkeley.edu/CIC/

"Supercomputing and Parallel Computing Research Groups" at the Computer Science department, Carnegie Mellon University:
http://www.cs.cmu/~scandal/research-groups.html

The Computer Architecture Home Page at University of Wisconsin, USA
http://www.cs.wisc.edu/~arch/www/jump.html

PART 3

THE SWITCH ARCHITECTURE

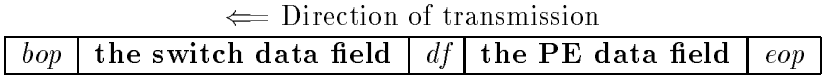
Chapter 7

The SWIPP packet and address format

This chapter describes and discusses the SWIPP packet format. This packet format is chosen to get small and dense switches and to support fast and independent establishment of new connections.

A *message* is the total amount of requested data forwarded by a user (i.e. application software etc.). This amount may be subdivided into smaller units more suited for transport by the network. These small units are called *packets*. In addition to the data to be transmitted the packet will contain a header with the information needed for routing the packet all the way to the destination. Many articles and textbooks use *frame* synonymously with packet. In this text, we will use "packet".

In our network a packet has two data fields: **the switch data field** and **the PE data field**. These two fields are enclosed and separated by three different unique control symbols: *bop* (beginning of packet), *df* (data follows) and *eop* (end of packet).



7.0.4 The switch data field

The **switch data field** contains information to and from the switches. This field is the main subject of this chapter.

7.0.5 The PE data field

The PE data field contains data transported from one PE to another PE. The switches do not control or correct the **PE data field**. Correctness etc. in this field depends entirely on end-to-end protocols performed between the Protocol Engines. The PE data field may contain any data. Thus, it may contain one or more ATM cells or one or more IP packets.

7.0.6 Packet size

The switch hardware architecture itself puts no regulations neither to packet size nor to the size of the PE data field. The only regulation given is that the PE data field has to contain an integer number of bytes.[1]

7.0.7 Control symbols.

All symbols have 9 bits. The symbols are divided into two groups: the data symbols with a leading 1 and the control symbols with a leading zero.

0	cccf	0000	Control symbol
1	dddd	dddd	Data symbol

The three control symbols used to enclose and separate the two data fields are:

bop:	beginning of packet:	used at the beginning of each packet.
df:	data follows:	used to separate the switch data field and the PE data field.
eop:	end of packet:	used at the end of each packet.

Another important unique control symbol is the *idle* symbol. This is a symbol which may be inserted anywhere in lack of other symbols. It can be inserted both inside and outside packets.

7.1 An overview of the switch data field

The default fields

Below we find the typical format of a packet header during transmission through the network.

bop	**The switch data field**				*df*
	(jc) jump counter	**(fal) forward address list**	**[d] dummy field**	**(bal) backward address list**	

We have the following 4 fields between the *bop* and *df* control symbols: the **jump counter**, the **forward address list**, the optional **dummy field** and the **backward address list**. Both the **jump counter** and the **dummy field** are 4-bit nibbles, while the **forward address list** and **backward address list** are sequences of 4-bit link numbers.

[1] Restrictions on packet size may for several reasons be practised by the Protocol Engines. Load, destination distribution and response requirements may give upper limits to packet size or requirement for fixed packet size. These restrictions may be maintained by the Protocol Engine hardware or software.

7.1.1 The jump counter (jc)

In front of the **switch data field** we find a nibble named the **jump counter**. The **jump counter** indicates how many more switches the packet has to pass. The number of remaining switches is equal to the number of unused address nibbles (length of the **forward address list**).

7.1.2 The address lists

In our address format the routing path all the way to the destination is described entirely by the source node. This kind of address format has many names where *source routing, self routing, hierarchic* and *relative* are the most used ones. The address lists are sequences of four bits (nibbles). Each 4-bit group gives the link number for one switch.

The forward address list is a sequence of output link numbers to be used later, while the backward address list is a list of the links already passed i.e. the link numbers in the opposite (backward) direction (in opposite order). The backward address list is created while the packet passes through the network. The number of the link on which a switch received a packet is put at the end of the backward address list in front of the *df*-symbol. The receiving PE may change the backward address list into a forward address list by putting the address nibbles in opposite order. The link number on which a switch receives a packet will be the output link number for a packet in the opposite direction.

In each switch the number of forward address nibbles is always decreased by one and the number of backward address nibbles is increased by one. Thus the total number of address nibbles in a packet is constant.

7.1.3 The dummy field

Data are transferred as symbols consisting of 8 bit of data and a leading one ("1"). Thus the nibbles in the switch data field have to add up to be an even number. If the number of nibbles without the dummy field is an odd number, the dummy field has to be included. If the number is even the dummy field is left out.

7.2 Examples of switch data fields

The shift, insertion and deletion of nibbles as a packet passes through a number of switches may look complicated. In this section we will give some examples to make it clearer. First we will describe how address nibbles are shifted, inserted and deleted from the address sequence. Then we look at some example **switch data fields** of different lengths.

7.2.1 Shift of address nibbles

Figure 7.1 gives an example of how the switch data field changes as a packet passes through switches. A packet is transmitted from a source (S) through 6 switches (i,j,k,l,m,n) to a destination (D). The numbers around the switches are the numbers of the input/output links. The contents of the **switch data field** are given in boxes.

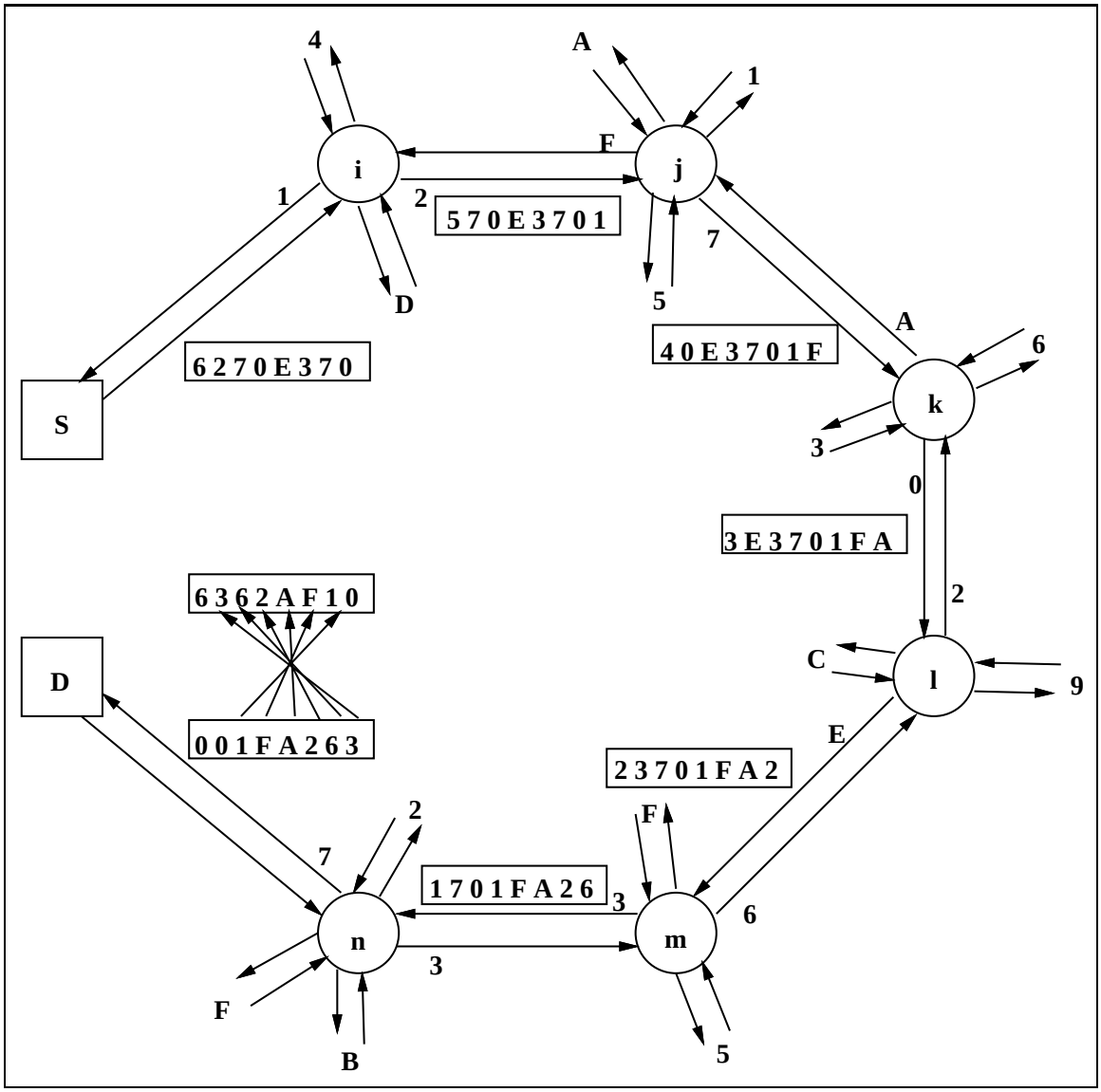


Figure 7.1: An example of address manipulation through switches.

The contents of the switch data field between switch k and switch l are given below.

The switch data field between switch k and l.							
jc	**forward address list**			dummy	**backward address list**		
	a_{l-m}	a_{m-n}	a_{n-D}		a_{i-S}	a_{j-i}	a_{k-j}
3	E	3	7	0	1	F	A
j	f	f	f	d	b	b	b

As the packet passes through the switches the switch data field changes:

	The switch data field							
After	meaning=value							
S	j=6	f=2	f=7	f=0	f=E	f=3	f=7	d=0
i	j=5	f=7	f=0	f=E	f=3	f=7	d=0	b=1
j	j=4	f=0	f=E	f=3	f=7	d=0	b=1	b=F
k	j=3	f=E	f=3	f=7	d=0	b=1	b=F	b=A
l	j=2	f=3	f=7	d=0	b=1	b=F	b=A	b=2
m	j=1	f=7	d=0	b=1	b=F	b=A	b=2	b=6
n	j=0	d=0	b=1	b=F	b=A	b=2	b=6	b=3

The lowercase letters have the following meaning:

j **jump counter (jc)**
f **forward address nibble**
d **dummy**
b **backward address nibble**

7.2.2 Shift of switch data nibbles

Each 'frame' in the tables in the remaining part of this chapter shows the first part of a packet. A packet sequence starts at the upper left corner, continues to the right, then from the left on the next line and so on. The next figure is drawn to illustrate the sequence.

n_0	n_1
n_2	n_3
n_4	n_5

⇐= Direction of transmission

n_0	n_1	n_2	n_3	n_4	n_5

Each small box contains one nibble. The smallest transferable symbol between nodes is a byte. For this reason the nibbles have been grouped two and two together in each row.

The frame on the left side in the following two figures is the packet at the source node. The frame on the right side of these figures shows the same packet when it has entered its destination. The letters indicating the kind of nibble follow the same syntax as given at the end of 7.2.1.

Example 1: 6 address nibbles

In the following example we have a packet with 6 address nibbles. From the source to the left it starts with 6 forward nibbles. At the destination it ends up with 6 nibbles in the backward direction.

Source:

bop	
j	f_0
f_1	f_2
f_3	f_4
f_5	d
df	

bop	
j	f_1
f_2	f_3
f_4	f_5
d	b_0
df	

bop	
j	f_2
f_3	f_4
f_5	d
b_0	b_1
df	

bop	
j	f_3
f_4	f_5
d	b_0
b_1	b_2
df	

bop	
j	f_4
f_5	d
b_0	b_1
b_2	b_3
df	

bop	
j	f_5
d	b_0
b_1	b_2
b_3	b_4
df	

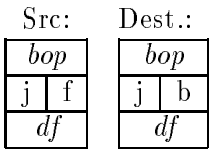
Destination:

bop	
j	d
b_0	b_1
b_2	b_3
b_4	b_5
df	

At the receiving end the path in the opposite direction can be created. We will not know whether the header field consists of 7 address nibbles or 6 address nibbles and one dummy nibble. For this reason the **jump counter** for the opposite direction must be sat to seven. The sequence of addresses in the opposite direction is as follows: b_5, b_4, b_3, b_2, b_1, b_0 and d.

Example 2: 1 address nibble

This smallest packet contains one address nibble.



Here it is obvious that the return address consists of one address nibble

7.3 Source routing and some other address formats.

7.3.1 Absolute addressing

With absolute addressing each destination has a unique identification. The switches have tables containing possible outgoing links for the different destinations. The tables may be bottlenecks when they have to be accessed for different packets simultaneously. The switches require circuitry to support loading of the tables. The tables are generated initially and it may be possible to update them during run-time. Depending on the number of destination nodes, it may be necessary to keep the tables in memory on separate circuitry. This may slow down the table-look-up time. The tables may contain several entries for each destination. This gives the possibility to re-route a packet if the primary path is occupied. The disadvantage of this is that packets from the same source to the same destination may be sent different ways and thus arrive out of order. This will make the receiving protocol more complicated.

7.3.2 Interval addressing

With interval addressing, each outgoing link on a switch covers a range of addresses. The destinations have a unique address depending on how they are connected to the switch nodes. A switch will compare the destination address of an incoming packet with the address ranges of the different outgoing links.

7.3.3 Source route addressing

The source-routed address format (like the one used by SWIPP) has the advantage that the packet carries the outgoing link numbers. Thus, there is no need for accessing a central address table. This both reduces the time to establish a connection and makes the transport of any packet independent of other packets routed for other outgoing links. The transmitting Protocol Engine has the responsibility for the entire packet path. The transmitting Protocol Engines must

discover traffic contention and use alternative routing paths if necessary. Since no address tables are required, the switch circuits will be faster and a smaller number needed for each switch.

Waste of address space in packet headers may be an argument against source routing. A closer study shows that the difference from absolute addressing is not large. This may be illustrated with two examples. Before the examples are given, we repeat that SWIPP may have maximum 15 address nibbles, which limits the address room.

First we find the maximum number of other hosts which can be reached from a basis host.This requires a tree topology where the "basis" host is on the root of the tree and the other hosts are at the leaves 15 steps away. The switch closest to the basis host is connected to 15 other switches. Each of these switches is connected to 15 other switches. This continues all the way for a total of 14 ramifications from the root. At the 15th forks from the basis host, the switches are connected to hosts. This is a topology with $2.9 \cdot 10^{16}$ switches and $4.37 \cdot 10^{17}$ hosts. With absolute addressing, 59 bits are required to identify the same number of hosts. This is one bit less than what the SWIPP source routing requires.

In our second example, only eight of the channels of each switch are connected to new switches. The other eight channels are connected to hosts. With this topology $9.04 \cdot 10^{11}$ switches connect $7.23 \cdot 10^{12}$ hosts[2]. For absolute addressing, 44 bits (11 nibbles) must be used to give an unique identification of all hosts. For source routing, the average address length from the basis host to all of the other hosts is a little below 15 nibbles. This is the case if a connection to every host is equally probable. If, on the other hand, the traffic pattern has a local characteristic, significantly shorter average address lengths can be achieved. Thus, the difference is not very large, and source-routing benefits from local communication.

7.4 A discussion of the use of backward address lists.

Some reasons for creating a backward address list are listed below.

• The implementation of the switch will be a little simpler when the packet length is not changing during propagation through the network. Without the **backward address list**, dummies have to be added and removed as the packet passes through the network. Now this is done only by the source Protocol Engine.

• Since it may be complicated to find the address of a communication partner at the first time of contact, it would be very helpful if the initiator generated the return address. For later use most Protocol Engines will store the addresses of their communication partner processes in memory.[3]

• If the receiver can not store a complete table of all possible return addresses, a free return address generated by the arriving packet will be advantageous. This is the case when the number of communication partners is higher than the number containable in the memory of the receiver.

• Another use is if some kind of signature is needed. The backward address list will give a clear confirmation of the identity of the transmitter.

[2]We here assume eight switches connected to each switch fifteen jumps away from the basis node. Due to the limitation to fifteen nibbles, the basis node can not address nodes at longer distances. If the last level instead had hosts only, the total number of host would have been $3.1 \cdot 10^{13}$.

[3]If the number of communicating partners is large or memory is expensive it may be preferable to store only a limited number of partner addresses in a cache-like table. Since communication is typically very conservative we would expect a high hit rate for most applications.

• For dynamic[4] networks an important use is as a tool for the Protocol Engines to make a map of their neighbours. Transmitting packets arbitrarily will be a necessary way to learn about connected switches and hosts.

• For network management a backward address list would be helpful for tracking errors.

[4] I.e. network without a fixed topology

Chapter 8

Overall architecture of the SWIPP switch

This chapter gives a top level functional description of the proposed implementation of the switch node. The designs have been done over several years. With the lay-outs made and some of the few years old technologies they have been implemented in, it is not possible to implement the switch on one circuit. Therefore the present description of the switch node is divided into different blocks. An overview of the different blocks and the connections between them is given. A discussion of the possibility to integrate the switch as one circuit is given in the summary chapter 21.

As described in earlier chapters, the Protocol Engines are connection points between the Computing Engine and the SWIPP interconnection network. This chapter will describe the switches constituting the core of the interconnection network.

8.1 The connections of the SWIPP switch.

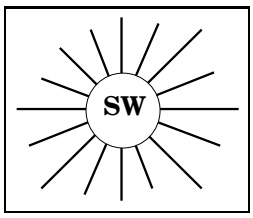


Figure 8.1: A switch has 16 full duplex links which can be used for connection to Protocol Engines or to other switches.

A SWIPP switch which has 16[1] links which can be connected to other net nodes is described. The other net nodes may be Protocol Engines or switches. Two switches can be connected by more than one link. All links are full duplex. Thus, data packets can be transferred in both directions simultaneously. Relative to a switch, each link can be divided into one input and one output link. Thus a switch with 16 links has 16 input links and 16 output links. The capacity of a link is 800 Mbits in each direction. A link may be implemented as two optical fibres or as a pair of electrical cables.

[1] For simplicity some figures have been drawn with 8 or 4 links. The right number of links is 16.

The switch elements and their connection wires/fibres may constitute any regular or irregular structure. New switches and links may be added to distribute the load and to reduce latency.

8.2 Top level functional description of the switch node.

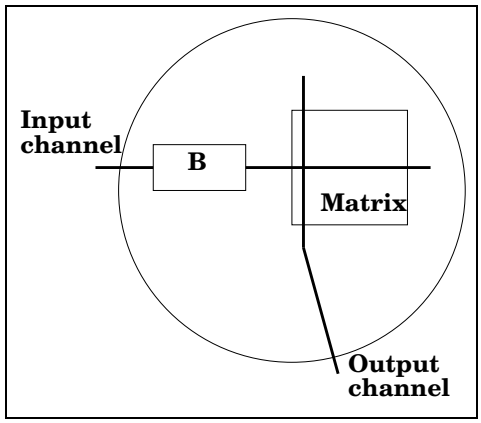


Figure 8.2: A switch has a total of 16 links where each link consists of an input link and an output link. The figure shows a connection between an input link and an output link.

Figure 8.2 gives a functional description of the switch principles. The functional blocks of one input link are drawn.

Each input link passes through a private FIFO buffer **B** on its way into the switch matrix. All incoming packets have to pass through these buffers. The part of the FIFO buffer closest to the buffer output is a pipeline where packet contents are interpreted and changed according to the packet protocol.

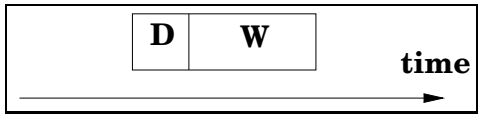


*Figure 8.3: The time a packet element spends inside **B**.*

Figure 8.3 illustrates the time a packet element spends inside **B**. This time can be divided into two parts, **D** and **W**, which are discussed in the following.

Minimum delay D to establish a connection.

The small delay **D** is the time used to establish a connection to an unoccupied output link. **D** is the time required by decoding and control circuits to extract the first output address from the packet head, to decode it and open the channel through the matrix. Hence the packet transmission from the source Protocol Engine to the destination Protocol Engine goes directly through the network with essentially no other delay than the hardware address decoding at each switch.

Storage in FIFO when output link is busy.

If the output link is available, a packet will be transferred through **B** without any other delay than **D**. If it is occupied, the data stream will be buffered in **B**, until the requested output link is ready and data can be forwarded.

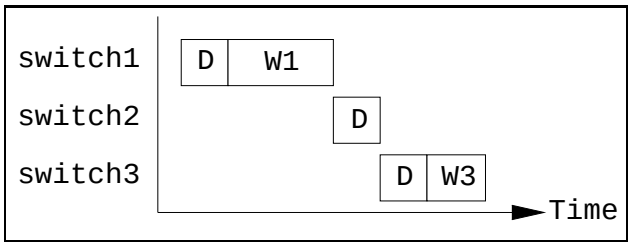


*Figure 8.4: Delays for a packet forwarded through 3 switches. In switch 2 the output link is unoccupied and the packet is only delayed the minimum time **D**. **W** is the waiting time due to queuing for an available output channel.*

Different parts of a packet may at the same time be buffered in a number of switches along its route. Large packets can fill up all buffers from the transmitting to the receiving Protocol Engine. Generally this is not efficient and global flow control should regulate the amount of unacknowledged data fed into the network.

8.3 The building blocks constituting the SWIPP switch.

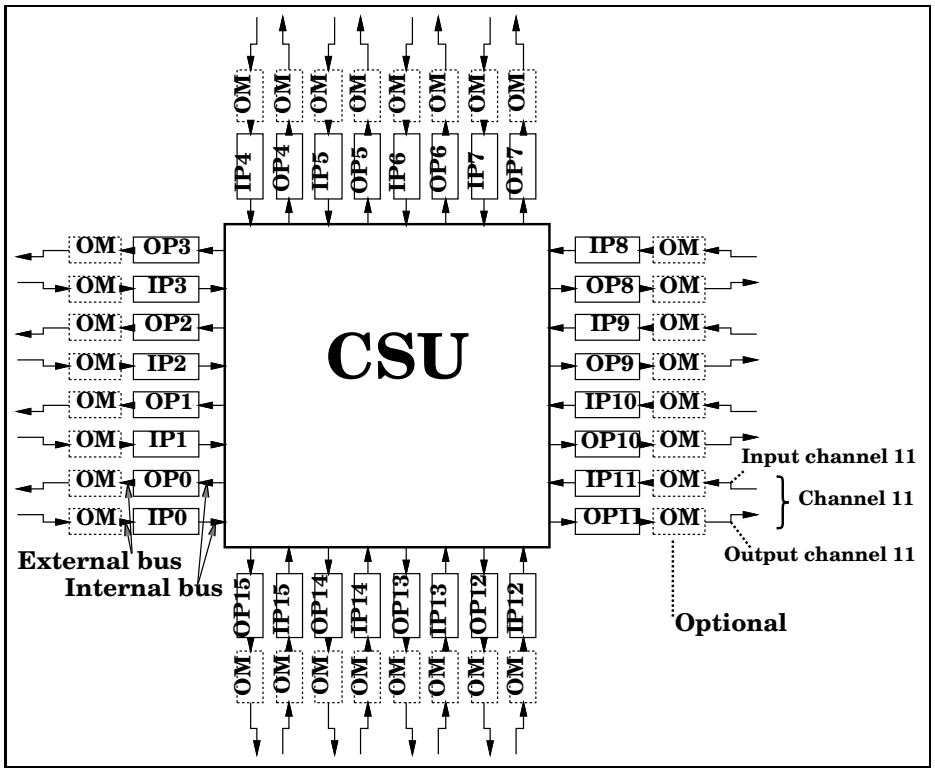


Figure 8.5: The figure shows a SWIPP switch with 16 links.

Figure 8.5 shows the main logical blocks of a switch. The **Central Switch Unit** (**CSU**) is drawn in the centre. Pairs of **Input Ports** (**IP**) and **Output Ports** (**OP**) are connected to the CSU. Outside the Input and Output Ports we find the **Optical Modules** (**OM**). The Optical Modules are optional and are required when the physical connections are implemented as fibre or coaxial cable.

Functions which can be executed independently of other channels are performed in the Input and Output Ports. The two main tasks of the CSU are to establish the physical connections and to perform arbitration between input links requesting the same output link.

Control information to establish and close connections is generated by the Input Ports and forwarded to the CSU through the private internal bus of each Input port. Data packets are kept in the Input Port until the path to the next net node is ready.

8.4 The external data buses.

The buses between the Optical Modules and the Ports are named external buses. These buses have a width of 9 lines. They are all unidirectional. The logical contents of these lines at a certain point of time is named a symbol. We have two kinds of symbols: Data symbols and control symbols. The leading bit distinguishes between these two possibilities. The signal definitions are explained in appendix A.

1	dddd	dddd	Data symbol
0	cccf	0000	Control symbol

A leading *one* indicates that the following 8 bits is a byte of data. A leading *zero* indicates that the symbol is a control symbol. The type of control symbol is identified by the three following bits. They are indicated as 'c's in the table above. The control symbol also carries one bit for the data stream in the opposite direction. This bit is the flow control signal marked by an 'f' in the table above.

8.5 The internal data buses.

The buses between the Ports and the CSU are named *internal buses*. These buses are 5 lines wide. All data buses are unidirectional: From Input Ports to the CSU and from the CSU to the Output Port.

9 bits from two succeeding 5-bit units make one data or control symbol. Notice the increase from three control bits in the 9-bit symbol to four control bits in the 5-bit symbol.

Positive clock level:	1	dddd	Data
Negative clock level:	f	dddd	symbol
Positive clock level:	0	cccc	Control
Negative clock level:	f	0000	symbol

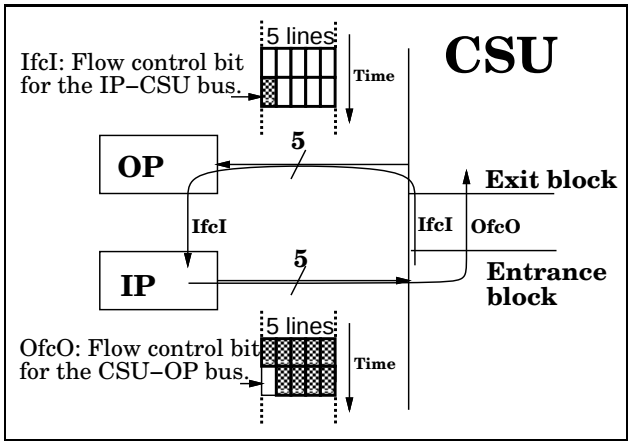


Figure 8.6: Data buses between the ports and the CSU for ***one*** *link. Flow control signals are transmitted through the bus of the peer port.*

The Input Ports generate control signals to the CSU concerning establishment and closure of connections. Flow control signals will give responses to these Input Port generated control signals. Since the data bus between the Input Port and the CSU is unidirectional the response signal has to be forwarded via the Output Port. This is the **IfcI**[2] signal illustrated in figure 8.6. The 10th bit in each group of two and two 5-bit units is used as a control signal for the opposite direction. This bit is indicated by an 'f' in the previous table. The outgoing data stream from the exit block

[2] Appendix A contains name syntax for flow control

to the Output Port also needs a flow control signal. This signal is named **OfcO**. In figure 8.6 the path of the **OfcO** is drawn from the Input Port via the entrance block to the exit block. This is described in more detail in chapter 9. The names of the flow control signals are given in Appendix A.

Chapter 9

The Central Switch Unit (CSU)

This chapter describes the logical architecture of the Central Switch Unit. Implementation details are found in chapter 16.

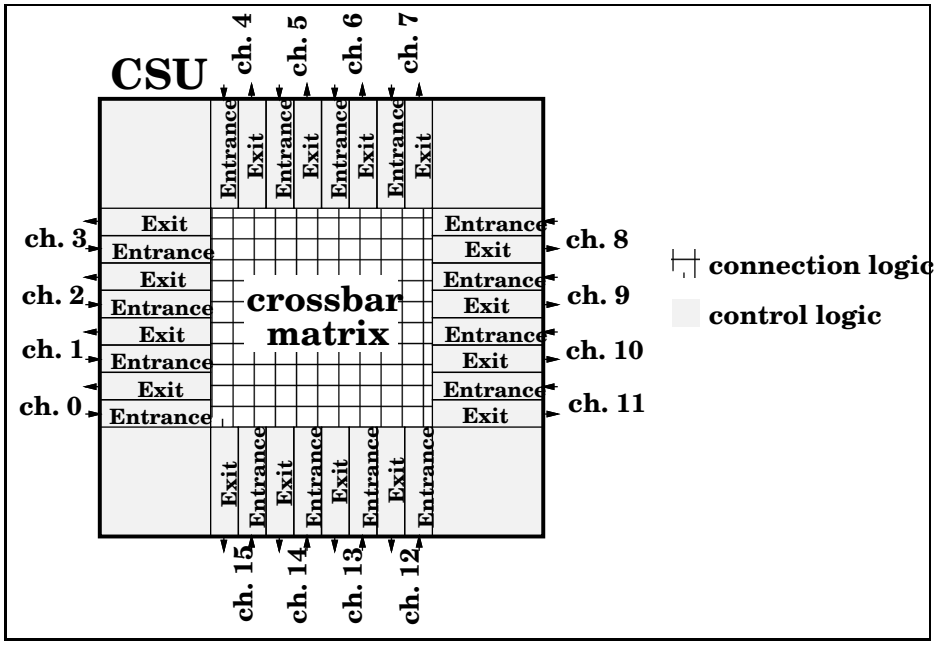


Figure 9.1: The logical blocks of the Central Switch Unit.

The Central Switch Unit can be divided into two parts: the connection block and the control logic.

The function of the connection block is to connect input channels and output channels. In SWIPP the connection block is a crossbar matrix. The connection pattern is determined by the arbiter part of the control block.

The control logic decodes and executes instructions received from the Input Ports, returns response signals, establishes connections, arbitrates between input channels requesting for the same output channel, and transfers flow control signals upstream from receiving to transmitting buffers.

9.1 The crossbar matrix

In a crossbar matrix wires from all input channels are connected through switch points to all output channels. This is illustrated in figure 9.2.

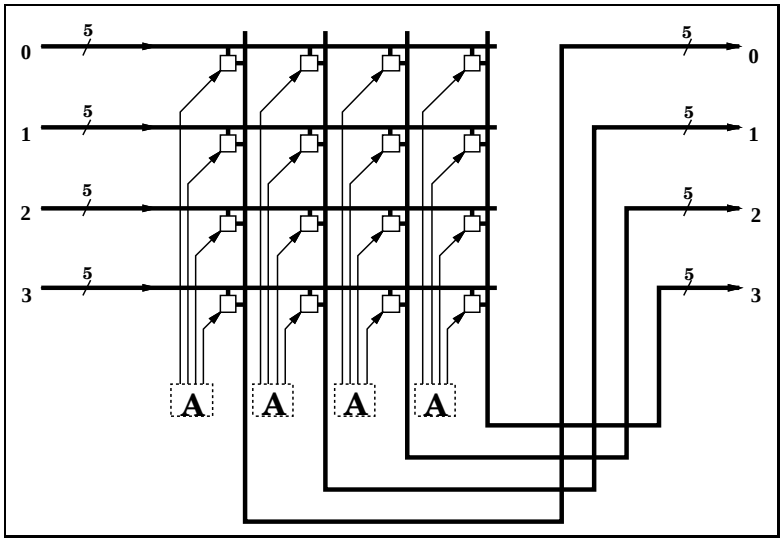


Figure 9.2: A logical description of a crossbar matrix. All input channels are connected to output channels through switch points (small boxes). All switch points belonging to a specific output channel are controlled by the same control block (A). For simplicity the crossbar has been drawn as a 4×4 *instead of* 16×16 *crossbar matrix.*

The non-blocking property.

In the crossbar matrix each pair of input and output channels has its own switch point. Thus a connection may be established between any free input and output channel independently of other connections already established.

Only one switch point between input and output channel.

In the crossbar circuit data have to pass only one cross point between input and output. A switch point can be made small and fast. With an increasing number of output channels, the transfer time will also increase because the capacitive load each channel represents will delay the function of the switch point. Thus, above a certain size an alternative would be to divide the switching into different levels. Another possibility is to choose the SWIPP implementation drawn in figure 9.3.

9.2 The control logic

The control logic is divided into 16 equal parts serving one channel each. Each of these parts can be divided into the **entrance** block mainly related to the incoming channel, and the **exit** block

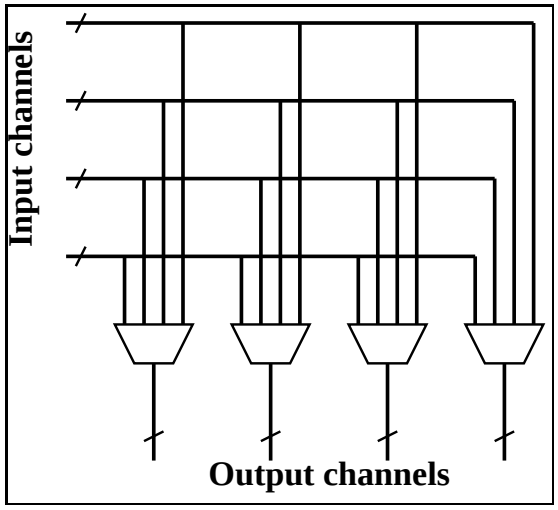


Figure 9.3: The SWIPP implementation of the crossbar matrix. Each output channel has one decoder choosing between all input channels. Due to small capacitive load this implementation has a higher clock rate potential than implementations based on transmission gates and clocked inverters. It also gives a better noise isolation to the un-connected inputs than an implementation based on the previous figure. For simplicity this crossbar has been drawn as a 4×4 *instead of* 16×16 *crossbar matrix.*

mainly related to the outgoing channel. A brief overview of the functions of the entrance and exit blocks will be given in the following.

9.2.1 The connections of the multicast entrance block

Figure 9.4 shows the connections of the CSU entrance block. On the left side we see the 5-bit input bus from the Input Port. Packet symbols and instructions for opening and closure of connections are received on this bus. The other connections are internal in the CSU. The lower right side shows the 5-bit packet bus to the crossbar matrix. Above the data bus we find the 16-bit request bus. This request bus is divided into one line for each exit block. On the upper right side we find the response bus which collects one line from each of the exit blocks. The remaining connections are the two single lines on top of entrance block. These are flow control lines to the peer exit block.

9.2.2 The connections of the multicast exit block

Figure 9.5 shows the connections of the exit block. On the left side we have the 5-line-wide internal data bus to the Output Port. Packet data are transmitted to the Output Port on this bus. The remaining connections are internal in the CSU. On the bottom we find the two flow control signals from the peer entrance block. On the lower right side we find the 16-line-wide request bus. This bus consists of one line from each entrance block. The 16-line-wide response bus has the opposite direction. It is divided into one line to each entrance block. A 5-line-wide data bus contains packet data from the crossbar matrix. On the top of the exit symbol we find the 16-line-wide control bus for the crossbar matrix. This bus has one line to 16 different crossbar

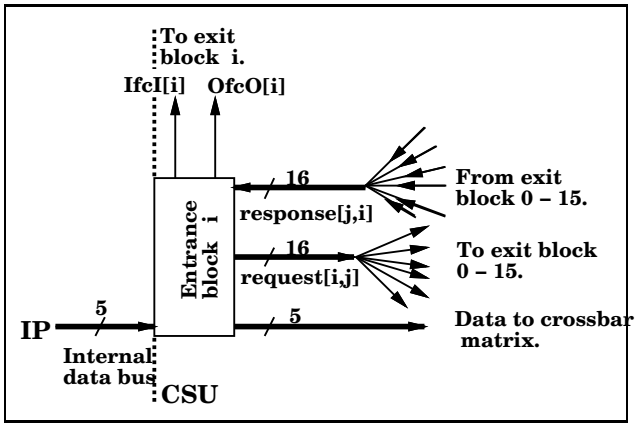


Figure 9.4: The figure shows the connections of the channel i entrance block of the CSU. A CSU has 16 entrance blocks, one for each channel.

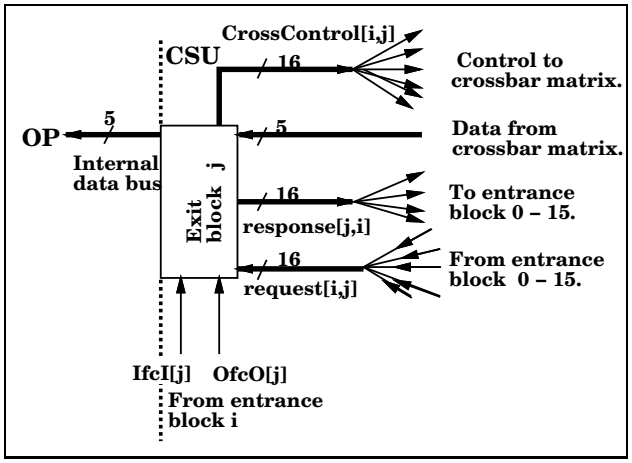


Figure 9.5: The figure shows the connections of the channel j exit block of the CSU. A CSU has 16 exit blocks, one for each channel.

points. These 16 crosspoints are the crosspoints between the input channels and output channel j.

9.2.3 Signalling between entrance and exit blocks.

There are two signals going between the entrance block and the exit block of the same channel. These signals are the flow control signals (**IfcI** and **OfcO**). They are introduced in subsection 8.5 and drawn in figure 9.4 and figure 9.5.

To establish connections, perform arbitration and forward flow control signals a complex network between entrance and exit blocks is needed. This network is described below.

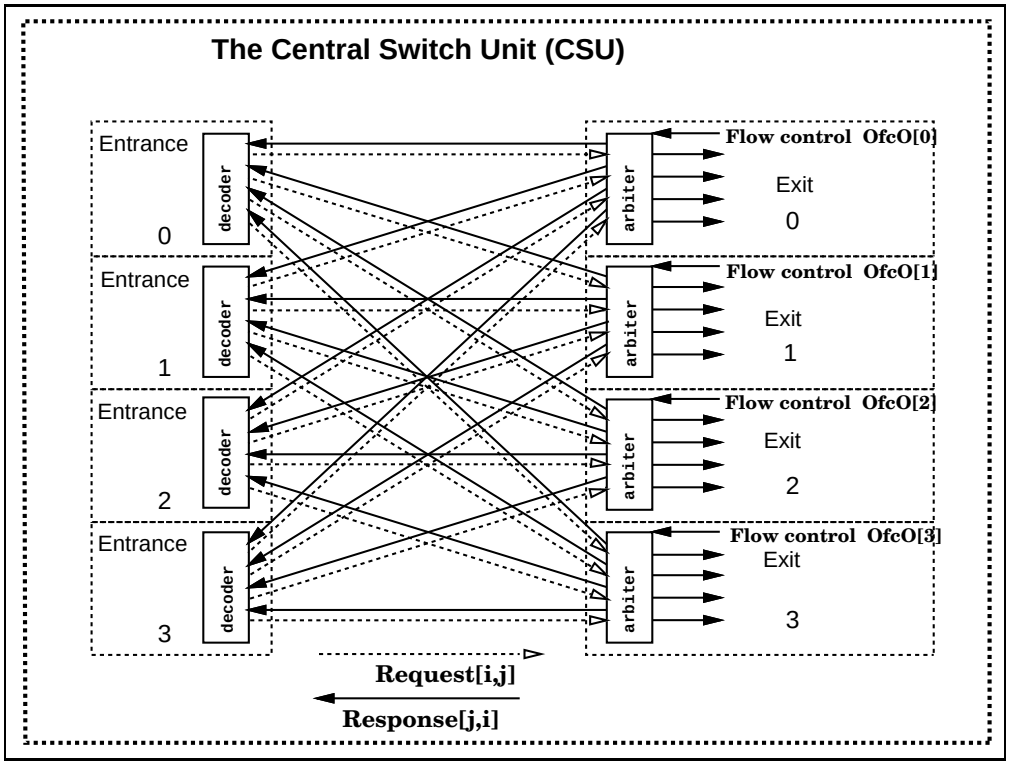


Figure 9.6: Signalling between the entrance and exit blocks. The request lines from the entrance blocks to the exit blocks are drawn as dashed lines with white heads. The response lines are drawn as solid arrows from the exit blocks to the entrance blocks. For simplicity this figure has been drawn with 4 instead of 16 channels.

The request lines

The entrance block receives 'request for connection' signals from the Input Port through the internal data bus. The address of the requested output channel is decoded by the entrance block into 16 bits and signalled on separate lines, one to each of the exit blocks. Each of these lines signal whether the input channel of the entrance block requests connections to that specific output channel of the exit block. One line between each pair of entrance and exit blocks give a total number for all combinations equal to $256 (16 \cdot 16)$ lines.

The response lines

An exit block has to transmit a response signal back to each entrance block. This signal will indicate whether or not the entrance block may forward packet data to that specific exit block. The response signal will be an AND function of whether the output channel is reserved for this input channel and whether the first buffer down the output channel is ready for more data.

9.3 The entrance block.

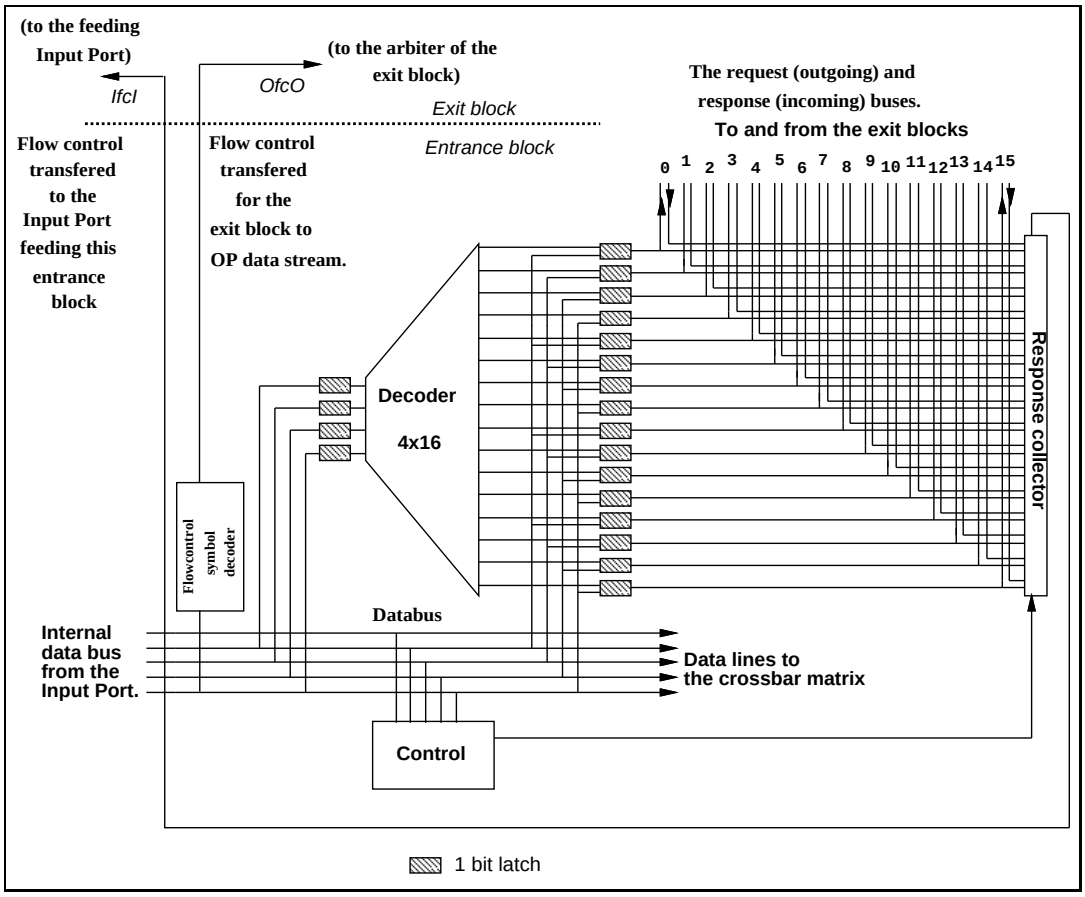


Figure 9.7: The entrance block of the CSU. ***IfcI*** *and* ***OfcO*** *are the flow control signals to the peer exit block. The 16 line pairs in the upper right corner are the lines between the entrance and exit blocks. Only 4 of these 16 pairs were drawn in the previous figure.*

Figure 9.7 shows the entrance block of the Central Switch Unit. On the bottom we find the control block. The control logic decodes some of the control symbols received on the incoming bus. The decoded control symbols are the request for unicast (*A4*), the request for selective broadcast (*A16*) and the close connection symbol (*eop*).

In the upper half we find the 4 to 16 decoder. On the left side of the decoder we find 4 memory latches. A unicast request to the entrance block will contain a 4-bit address which is loaded into these 4 memory cells. The 4 bits will propagate through the decoder to the 16 memory cells on the right side. With a unicast request one of the 16 cells will be active while the remaining 15 are passive.

A selective broadcast request to the entrance block will contain an address bit map with 16 bits. These 16 bits will be written directly to the 16 memory cells on the output of the decoder. In this case any number of these memory cells can be active.

On the right side of the figure we see that the output lines from the 16 cells are routed one to each of the exit blocks. Each of the entrance blocks also receives one line from each of the exit blocks. The original lines from the 16 cells and the 16 response lines from the exit blocks are

both routed to the response collector.

The **response collector** is a block generating the flow control signal for the incoming data stream. In the case of a unicast request the output of the **response collector** will go high when the response signal from the requested exit block goes active. If we have a multicast request the 16 request lines will be used as a bit mask. The **response collector** output will be the result of a logical AND function of the response lines from the requested exit blocks. The response lines from the exit blocks not requested are ignored. As the requested output lines get ready for the input line, they will be kept reserved until all requested outputs are available. Thus the output lines first ready will stay idle until the last requested line is available. When all are ready the packet will be forwarded to the output lines simultaneously. The flow control signal will be routed back to the Input Port via the exit block and the Output Port. It will request the Input Port to start or stop transmission of packet data.

The **close connection** control symbol (*eop*) will erase the 16-bit register thus making all output cells passive. It will also disable the response collector.

We have two flow control signals in figure 9.7 which are forwarded to the exit block of the channel. The output signal from the response collector, **IfcI**, has already been described. The other flow control signal, **OfcO**, belongs to the data stream through the exit block and the Output Port. It is received on the data bus from the Input Port and forwarded to the exit block arbiter of the channel.

9.4 The exit block.

The main element of the exit block is the arbitration logic. The 16 request lines enter the arbitration block from the left side. The result of the arbitration leaves this block on the right side and is used directly to control the switch points of this output channel. A logical AND is performed between each of the 16 arbitration outputs and the flow control signal **OfcO**. The results of these 16 AND functions are the response signals which are routed back to each entrance block.

Each exit block receives two flow control signals from the peer entrance block. As stated above the **OfcO** flow control signal is used for the response signals. The **IfcI** flow control signal, which is the flow control signal of the entrance block **response collector**, is routed via the local output multiplexer to the Output Port.

9.4.1 A connection example.

Figure 9.9 visualises some of the functions of the Central Switch Unit. Only the relevant logic has been drawn.

In figure 9.9 a request for output channel 4 enters entrance block 0 from Input Port 0. The request is decoded and transmitted on the line from entrance block 0 to exit block 4. Exit block 4 performs the arbitration between this and possible other requests. The result of this arbitration will be combined with the flow control signal concerning the down stream buffer status. This flow control signal is shown as a dotted line through Input Port 4 to the arbiter (**A** in figure) of exit block 4. In the arbiter the signal is combined with the arbitration result and routed back to the entrance block. From the entrance block it is forwarded back to the Input Port via the exit

block and Output Port of the same channel. (The figure shows the data path as a dashed line from Input Port 0, through the crossbar matrix of the CSU, to Output Port 4.)

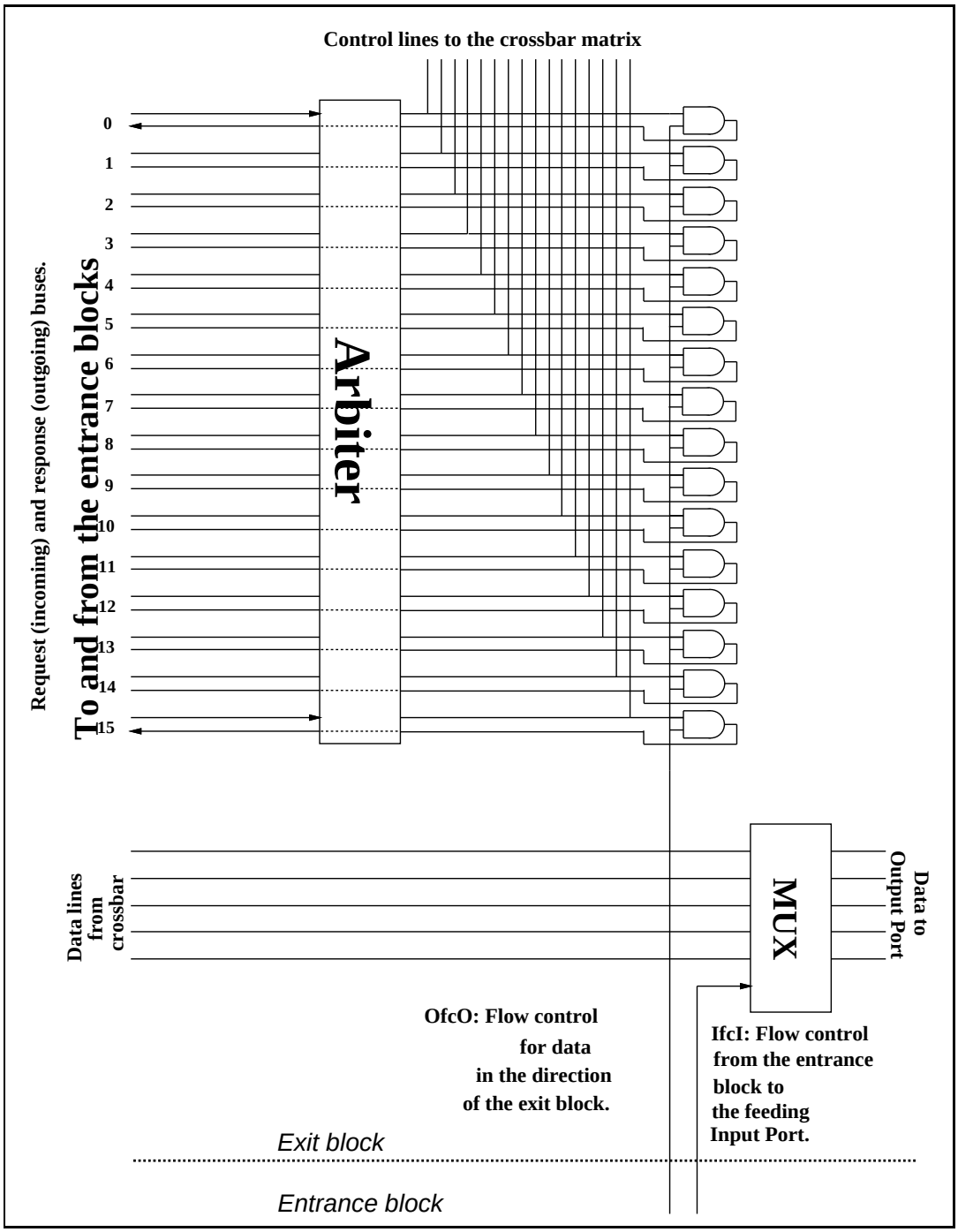


Figure 9.8: Exit block of the CSU.

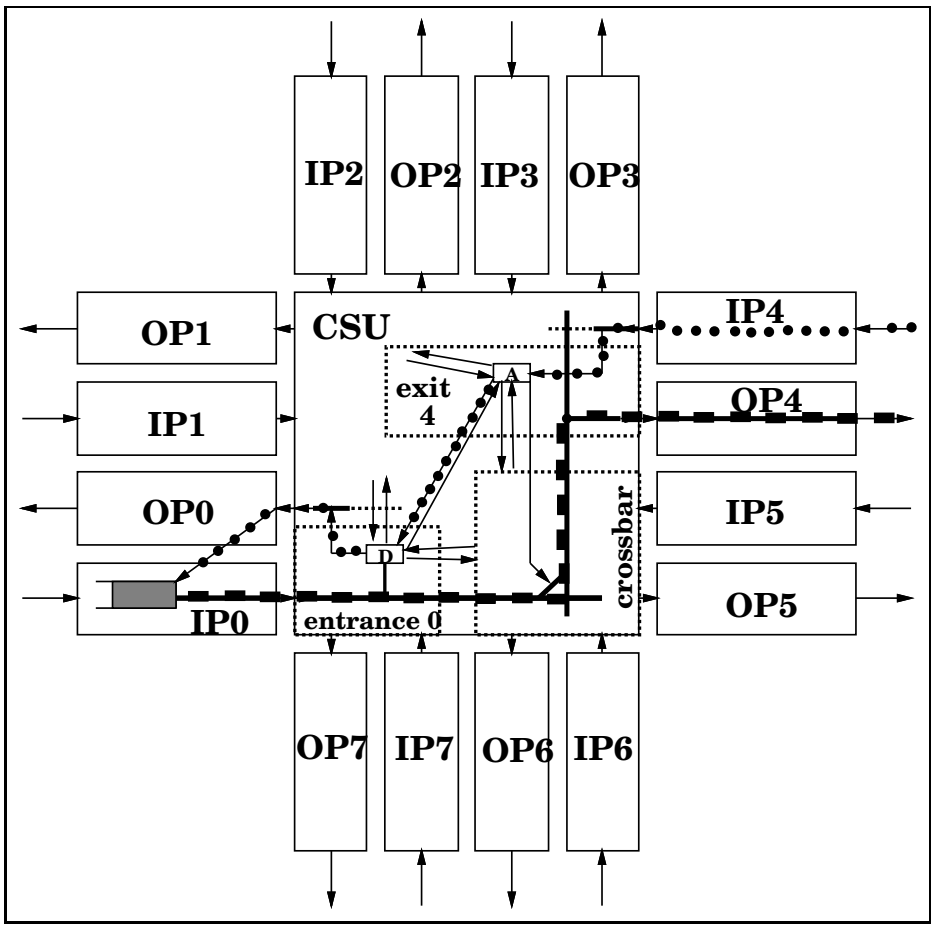


Figure 9.9: The figure illustrates some of the functions of the Central Switch Unit. To simplify the figure it has been drawn with 8 instead of 16 channels.

Chapter 10

The Input and Output Port

This chapter describes the function of the Input and Output Ports.

The ports perform those of the switch functions that can be placed outside the CSU. A switch (or Protocol Engine) has one pair of Input and Output Ports for each channel.

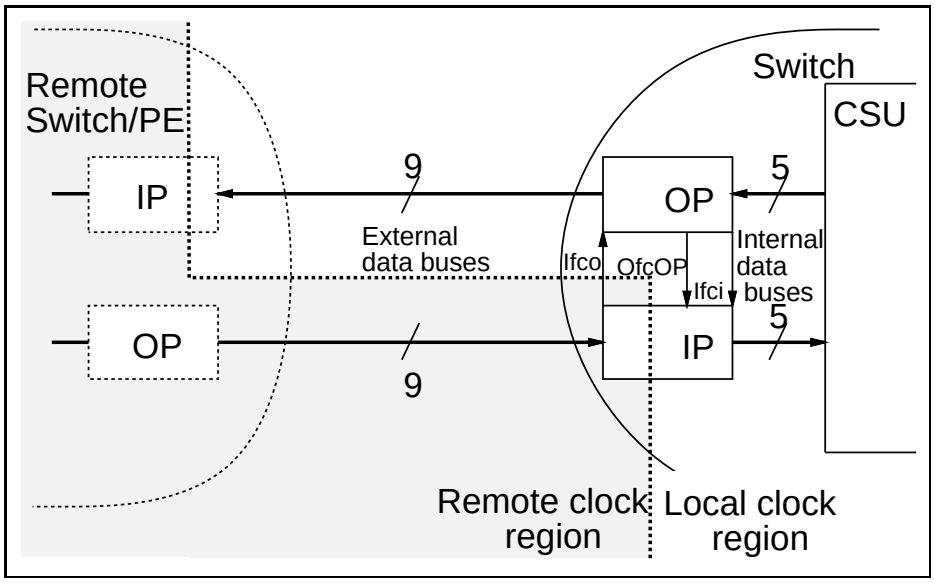


Figure 10.1: The Input and Output Ports of a channel between two network nodes. Due to speed limitations, a direct connection like the one drawn in the figure can only be used when the distance is short. For longer distances, more logic, such as a parallel-to-serial converter for serial transmission over coaxial cable or fibre is required.

A channel between two network nodes requires two Input Ports and two Output Ports. One Input Port and one Output Port are needed for each direction. In Protocol Engines, reduced versions of the Input and Output Port are used.

10.0.2 The interconnections of the Input and Output Port.

The Input Port (the lower right in figure 10.1), has a 9-line-wide input bus, a 5-line-wide output bus, and one outgoing and two incoming flow control lines. The input bus is the external bus from the Output Port at the remote net node while the output bus connects the Input Port to the Central Switch Unit (CSU). The flow control lines go between the Input Port and the peer Output Port.

The Output Port (upper right in figure 10.1) has a 5-line-wide input bus, a 9-line-wide output bus, and the three (one incoming and two outgoing) flow control lines to the peer Input Port. The 5-line-wide input bus is from the CSU while the 9-line-wide output bus is the external connection to the Input Port at the remote net node.

Two of the three flow control signals between the Input and Output Port concern the incoming data stream. **IfcI**[1] is a signal from the entrance block routed via the exit block and the Output Port. **IfcO** is a flow control signal generated in the Input Port. It is transmitted to the Output Port where it is inserted into the outgoing data stream. The last flow control signal, **OfcOP** concerns the outgoing data stream. It is generated in the Output Port. These flow control signals will be discussed further in chapter 12.

10.1 The Input Port.

10.1.1 The function of the Input Port

A short overview of the main functions performed by the Input Port is given below.

Reading and updating of address format.

The second byte of a packet contains the output channel number. This byte is read by the control logic of the Input Port. The contents of the address field are changed as the packet is transferred through the switch, as described in chapter *7, The SWIPP packet and address format.* This changing of the address sequence is executed by the Input Port.

Creating instruction symbols for the CSU

The Input Port creates instruction symbols which are forwarded to the switch circuit. These symbols are used to establish and close connections.

Storage of incoming data

Another important function is delay of incoming data. The Input Port will read the address field of an incoming packet and forward a request for connection based on this. All the time until a positive flow control signal is received from the CSU, new packet data may arrive. The Input Port stores these incoming data. The time before a positive response is received is short if the

[1] Appendix A contains name syntax for flow control signals

output channel is ready and may be long if the output channel is busy and several input channels request for the same output channel.

Clock border

All switches operate on their own locally generated clocks. Since switches are connected, there must be some place where circuitry transfer signals from one clock rate and clock phase to another clock rate and phase. This border between clock regions is in the FIFO buffer inside the Input Port.

Error detection and fault recovery

It will not be possible with reasonable means to guarantee that errors which influence the transmission of packets through the network will not occur. Errors may arise due to software faults[2], hardware faults[3] or noise[4]. The consequence of errors may be small, like when a packet is routed to a wrong destination. In the more serious cases we may have deadlocks where parts of the network are blocked so that no data in these parts can be transferred. When this happens, one solution may be to reboot the network. Thus, also packets in parts of the network not affected the first time will be lost. It will take time for the sources to discover the loss of these packets and to retransmit them. Due to this, we may want to add some additional logic to reduce the need for rebooting. The solution chosen for SWIPP is to allow the Input Ports to drop packets when no data have been propagated for a certain time period. This programmable period has to be long enough to prevent that packets are lost just because of a short queue for the output channel. A possible time-out period is 50 ms.

Errors and error treatment are discussed in more detail in chapter *14, Error handling*.

In SWIPP, a time-out will result in that a disconnect (*eop* end-of-packet) symbol is generated by the Input Port and forwarded to the CSU. Since the CSU has no input buffering the disconnect symbol is executed immediately. The Input Port will also throw away the remaining part of the packet.

10.1.2 The main blocks of the Input Port.

Figure 10.2 shows a sketch of the different blocks of the Input Port.

Clock regions

Two blocks transfer symbols/signals from the region of the transmitting clock to the region of the local clock. These blocks are the elastic FIFO and the elastic register.

[2]Combinations not covered or other faults in protocols, routing algorithms, etc. due to lack of understanding and overview by the designers.

[3]As the previous, but in hardware.

[4]Noise due to electomagnetism, mismatch of components, ageing etc. These faults are known but expected to be rare and to expensive to be prevented completely.

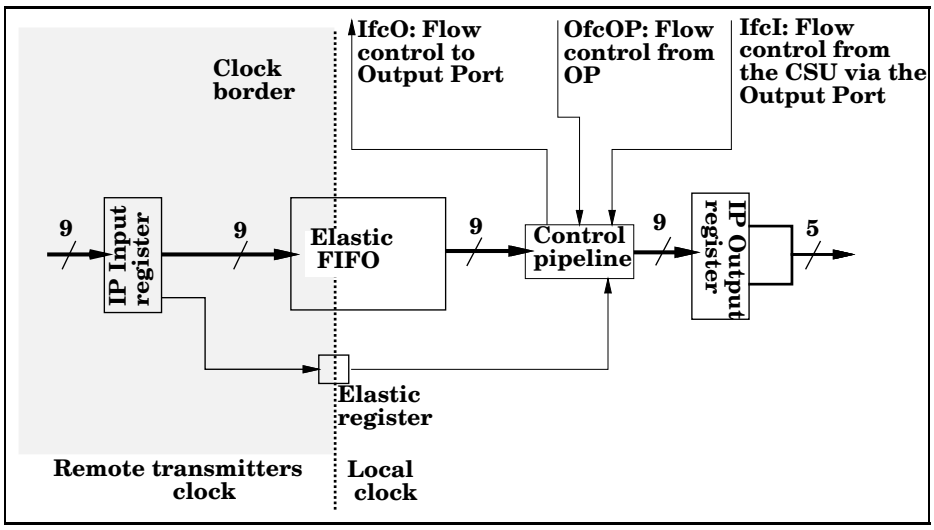


Figure 10.2: Main blocks of the Input Port. The clock border mentioned above is illustrated as a dashed line in the figure. The clock region on the left side, coloured grey, is clocked by the clock of the transmitting node. The white region on the right side is clocked by the local clock.

The IP input register

The incoming data stream contains both packet data and independent control information. The function of the **IP input register** is to decode the incoming symbols into two directions: data and control symbols belonging to the packet stream are forwarded to the elastic FIFO, and control information independent of the packet stream is forwarded to the elastic register. An example of the last group is the flow control signal for the data stream in the other direction.

The elastic FIFO and the elastic register

The purpose of the **elastic FIFO** and the **elastic register** is to transfer data consistently from one clock region to the other. This has to be done carefully to diminish the chance of reading metastable states. This is discussed in more detail in the implementation chapter *18, Implementation of the elastic FIFO*

The elastic FIFO

The elastic FIFOs of the Input Ports are the main storage facilities of the interconnection network. They store packet data and control symbols. A number of factors influence on the minimal and optimal buffer sizes. The buffer at least has to be large enough to store incoming symbols until a connection to an unoccupied channel has been established. Another minimum requirement comes from the flow control scheme we are using. It is essential that incoming data do not exceed the capacity of the FIFO. This is discussed further in chapter *12, Flow control and input buffering.* Traffic analyses may indicate that some buffer lengths give more efficient networks than others. This is the subject of chapter *22, Traffic modelling of an input buffered switch.*

The elastic register

The function of the **elastic register** is to transfer a flow control signal from the region of the transmitting clock to the region of the local clock. The flow control signal is read from the input bus of the Input port and forwarded to the Input Port **control pipeline**.

The IP output register

The function of the **output register** is to concentrate the 9-bit symbols into a data stream with double bit rate per line and half bus width compared to the internal format of the Input Port. This is done with a five-bit register which transfers the upper 5 bits of the symbols during the high clock level and the lower 4 bits plus a flow control bit during the low clock level.

The control pipeline

The **control pipeline** performs the following functions:
- Changes packet contents and symbol sequence according to the address protocols,
- Generates instruction code for the CSU about establishing and closing of connections, and
- Forwards packet symbols to the CSU.

The pipeline consists of a number of state machines, each handling a part of the pipeline.

10.1.3 Example of a symbol sequence transferred to the CSU.

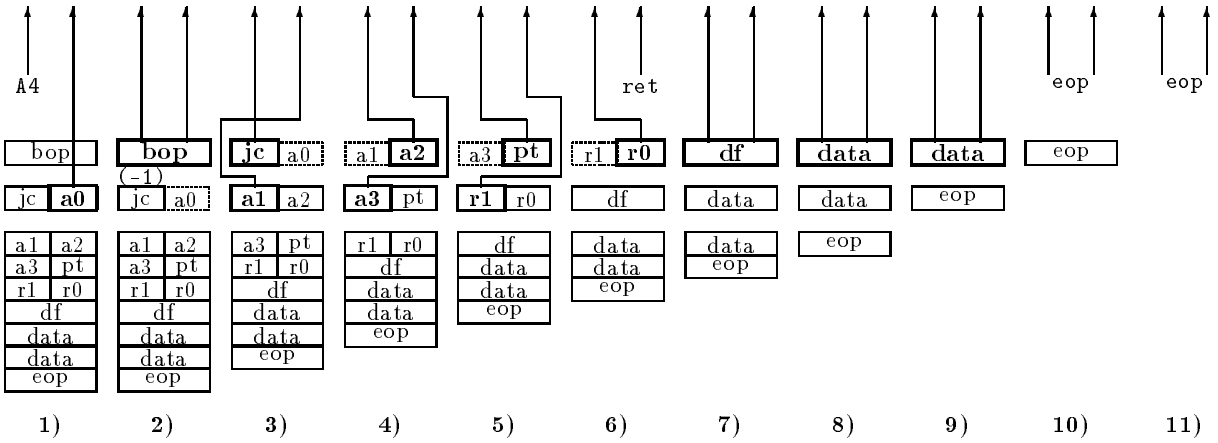


Figure 10.3: This figure shows an example of how symbols are transferred to the CSU. The 11 columns show 11 time steps, beginning at the left. The output to the CSU is at the top. Below is the control pipeline with the FIFO buffer at the bottom. The two first bytes are drawn a little above the others. The pairs of arrows above each column show the parts making a new symbol. The left arrow in each pair represents 5 bits, while the right represent 4 bits.

To make it easier to understand how packets are forwarded we will give an example. Figure 10.3

shows how a packet is interpreted and forwarded to the CSU. Each column represents a time step.

In step 1 a valid *bop* (beginning-of-packet) symbol has been detected in the first word and a valid data symbol in the second byte. A symbol requesting for an output channel is created and forwarded to the CSU. It contains a fixed part (`A4`), and the least significant part of the second byte, which contains the address of the first output channel.

After step 1 the Input Port will wait for a positive flow control signal as a response to its request for connection. In the meantime `idle` symbols will be forwarded to the switch circuit. This waiting step is not shown in figure 10.3.

In step 2 a positive flow control signal has been received from the CSU. The *bop* symbol is transmitted to the CSU and the remaining contents of the FIFO are shifted forwards.

Step 3 shows how a new symbol is created from two halves. The `a0` symbol is removed from the packet by not copying it to the CSU.

In steps 4 and 5 symbols are created from the lower part of the first byte and the higher part of the second byte.

In step 6 the return address is inserted at the end of the address field.

Steps 7, 8, 9 and 10 show how the remaining part of the packet is forwarded to the CSU.

The *eop* is used as a command to terminate the connection. This signal is repeated until a negative flow control signal is received from the CSU. This is to make sure that a new packet will not use an old positive flow control signal.

10.2 Output Port

The main blocks of the Output Port is drawn in figure 10.4. The functions of the Output Port is to extract flow control signals at the input bus and to insert flow control signals into the packet stream at the output bus.

Extraction of flow control signals at the input bus

The 5-line-wide input bus from the CSU is decoded into a 10-line-wide local bus. One of the 10 lines is the flow control signal. The flow control line is routed to the Input Port.

Insertion of flow control signals into the data stream at the output bus

The Output Port receives a flow control signal from the Input Port. This signal is merged into the outgoing data stream. The flow control signal is transferred in a specific bit in all control symbols.

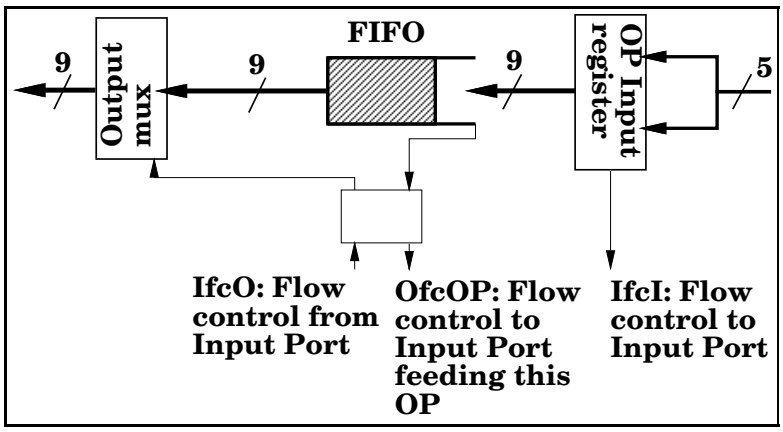


Figure 10.4: Main blocks of the Output Port. A simple decoder at the incoming bus (part of OP Input register) separates the flow control from the data stream. A FIFO is used to store incoming data when flow control signals are inserted into the outgoing data stream. The flow control signals are inserted if the FIFO capacity has not been exceeded. If the FIFO is full a flow control signal (OfcOP) is transmitted upstream to the Input Port feeding the Output Port drawn.

Chapter 11

The Optical Module

This chapter gives a logical description of our optical modules.

The connections between net nodes can in each direction be implemented as:
- a wide electrical cable consisting of several electrical wires.
- an electrical cable (coaxial cable) where all data are transmitted serially on one wire and
- an optical fibre where all data are transmitted serially[1].

Optical fibres may be preferred to electrical wires to achieve: higher bandwidth, shorter propagation time, larger noise resistance or smaller physical size.

For encoding and decoding of data for serial lines (fibre or coaxial cable) there are a number of chip sets commercially available. Some of these have a lower clock rate than what SWIPP has been designed for. Thus, SWIPP has to run on a lower clock rate if these chip sets are chosen.

An encoding and decoding architecture for serial lines has been investigated for SWIPP. The architecture is based on the Conditional Invert Master Transition (CIMT) scheme. This scheme and a more thorough discussion of optical communication than what will be presented in this thesis, are given in two papers by this author and fellow researchers [38] [66]. We will in this chapter give a brief overview of our architecture.

The SWIPP standard symbol interface offered by the net nodes.

0	$c_0 c_1 c_2 f$	0000	Control symbol
1	$d_0 d_1 d_2 d_3$	$d_4 d_5 d_6 d_7$	Data symbol

The symbol format given above is used for short parallel electrical cables between net nodes. When optical fibres are used, this symbol format is used between the Optical Modules and the remaining part of the net nodes. Another format is used for the fibres between the Optical Modules.

[1] There are also some more exotic alternatives like one optical fibre with different frequencies (colours) for each bus line.

11.1 The chip set for the SWIPP Optical Module.

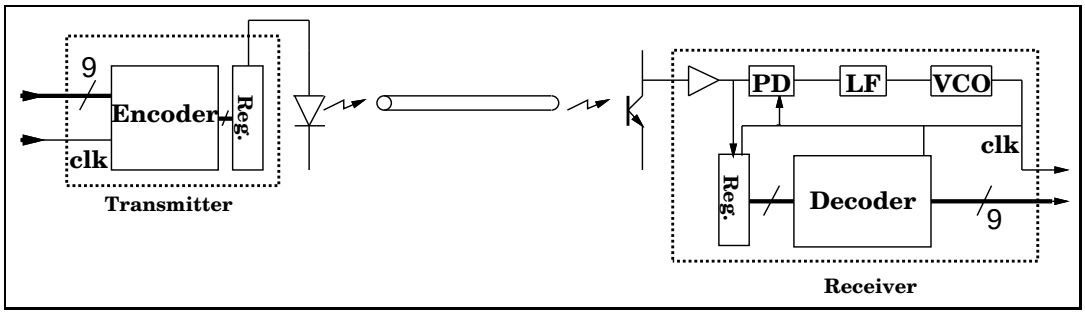


Figure 11.1: Block diagram of the SWIPP Optical Module.

Figure 11.1 shows the SWIPP Optical Modules. The transmitting part consists of one ASIC[2] and one discrete component: the light source. The light source may be a LED[3] or a LASER[4]. The receiving part consists of a discrete photo transistor and an ASIC.

The system consisting of the transmitter, receiver and code has to be selected in such a way that the receiving part can perform efficiently :
1. clock recovery/bit synchronisation,
2. word synchronisation, and
3. frame/packet synchronisation.

The first requirement is fulfilled by the analogue part of the receiver. The analogue part consists of a Phase Locked Loop (PLL). In figure 11.1 the Phase-Locked Loop consists of the PD (Phase Detector), the LF (Low pass Filter) and the VCO (Voltage Controlled Oscillator).

The word format consists of two token bits and eight data bits. The token bits are either 01 or 10 so that there will always be an edge in the middle. This edge is used by the PLL.

Word synchronisation is attained by an initial training sequence. The training sequence is a repetition of the word 0111110000. During the training period the PLL will try to lock at the increasing edge of the sequence. After the training sequence has finished the last 8 bits will contain data. The word synchronisation relies on bit synchronisation i.e. that the PLL keeps the edge between the 01 (10) token pair.

Packet synchronisation is attained by specific control words in the beginning and end of each packet.

Due to the nature of the receiving photo transistor it is also necessary that the code has low DC value i.e. the difference between the numbers of zeros and ones should be as small as possible. This is done by transmitting words inverted if they would otherwise increase the difference between ones and zeros. The following table shows how this is done.

[2]Application Specific Integrated Circuit
[3]Light Emitting Diode
[4]Light Amplification by Stimulated Emission of Radiation

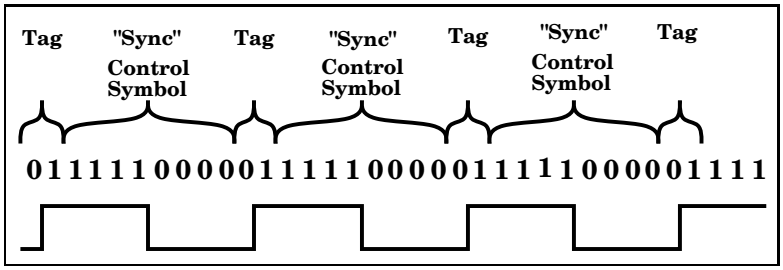


Figure 11.2: Training sequence to attain word synchronisation.

Previous words	Present word	Tag	Data part		
$\#\,1 > \#\,0$	$\#\,1 > \#\,0$	10	$\overline{d_0}\,\overline{d_1}\,\overline{d_2}\,\overline{d_3}$	$\overline{d_4}\,\overline{d_5}\,\overline{d_6}\,\overline{d_7}$	inverted data
$\#\,1 \geq \#\,0$	$\#\,1 \leq \#\,0$	01	$d_0 d_1 d_2 d_3$	$d_4 d_5 d_6 d_7$	non-inverted data
$\#\,1 \leq \#\,0$	$\#\,1 \geq \#\,0$	01	$d_0 d_1 d_2 d_3$	$d_4 d_5 d_6 d_7$	non-inverted data
$\#\,1 < \#\,0$	$\#\,1 < \#\,0$	10	$\overline{d_0}\,\overline{d_1}\,\overline{d_2}\,\overline{d_3}$	$\overline{d_4}\,\overline{d_5}\,\overline{d_6}\,\overline{d_7}$	inverted data
	$\#\,1 = \#\,0$	10	$c_0\,c_1\,c_2\,f$	$\overline{c_0}\,\overline{c_1}\,\overline{c_2}\,\overline{f}$	control symbol

Notice that the combination inverted tag (10) and equal number of ones and zeros is not needed for transmission of data. This combination is used for control symbols like the symbol used in the training sequence.

11.1.1 The encoder

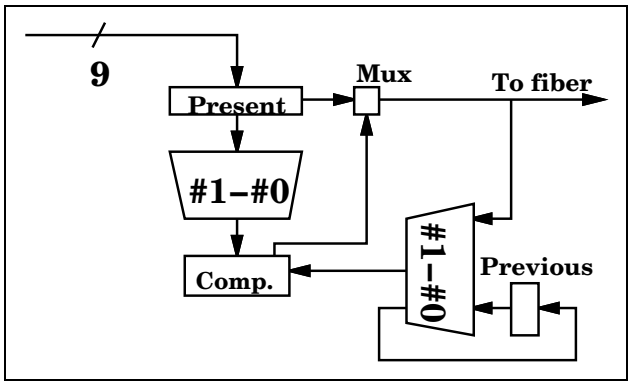


Figure 11.3: Block diagram of the SWIPP encoder.

Figure 11.3 shows the encoder of the SWIPP system. The encoder consists of two counters which calculate the difference between the numbers of ones and zeros. One counter calculates for the present word while the other calculates for the previous words. The signs of the output values of these counters are used by the comparator to control the multiplexer. The multiplexer decides whether the data should be transferred inverted or non-inverted.

11.1.2 The decoder

The main function of the decoder is to forward the received data words inverted or unchanged depending on the tag.

11.2 Methods and material for transmission between net nodes.

11.2.1 General: Parallel and serial transmission methods.

In "parallel" transmission, two or more bits are transmitted simultaneously. One "physical transmission channel" (wire, fibre, frequency or light-wave length) is required for each parallel bit. In most implementations, one of these physical channels contains the missing clock information. The disadvantage of the parallel transmission method is the large space required for the wires. A problem may also be signal skew between different signal wires.

In serial transmission all bits are transmitted serially on one physical channel. Clock information has to be incorporated into the data stream. One disadvantage of the serial transmission method is the additional circuitry required at the receiving end for recovery of the clock signal. A single serial wire must have a bandwidth n times as large as n wires used in parallel. Thus as the total bandwidth demand increases the serial line will meet electrical (or optical) bandwidth limitations earlier than the parallel wires.

11.2.2 Transport medium.

The commonly used alternatives for connection between net nodes are:
- Unshielded, single wire,
- Twisted pair,
- Coaxial cable, and
- Optical fibre.

For higher bandwidths the main limiting parameters are:
- Energy loss in the transport medium,
- Noise from other signal sources,

and for optical fibres also:
- Different signal propagation time due to different signal paths through the transport medium, and
- Different signal propagation time for different signal frequency components.

Single, unshielded wires and twisted pairs.

Single, unshielded wires can be used for bandwidths up to 1 MHz for a few meters. Twisted pairs of wires have demonstrated bandwidths up to 250Mbps with reduced voltage swing. To achieve total bandwidths of several hundred Mbps a number of single lines or preferably a number of twisted pairs are used in parallel.

Type	Attenuation pr. 100m					Price (USD) pr. km
	30MHz	100MHz	200MHz	500MHz	1GHz	
M17/119-RG174/U50Ω	16dB	30dB	45dB	73 dB	105,3dB*	667,-
H100 50 Ω	2,3dB	4,5dB	6,5dB	10,2dB	14,3dB*	2402,-

* denotes estimate by the author based on the other numbers.

Table 11.1: Technical data for coaxial cables from several suppliers have been collected. The table shows the coaxial cable with the lowest price and the cable with the best attenuation.

Type	Light wave length	Attenuation	Bandwidth distance	Price(USD)/km
ELFA 55-955-25	850nm	$\leq 3,5$dB/km	$\geq 160MHz \cdot km$	2640,-
	1300nm	$\leq 1,0$dB/km	$\geq 500MHz \cdot km$	

Table 11.2: Example of low price optical fibre. The fibre is multimode graded-index.

Coaxial cable and optical fibre.

For serial transmission at high bandwidths coaxial cable or optical fibre has to be chosen. In the following, some examples of coaxial cables and optical fibres are presented. Technical data and prices have been collected from some of the major Norwegian suppliers of electrical components and cables.

Table 11.1 shows two examples of coaxial cables. A coaxial cable may be regarded as a low-pass filter with increasing attenuation for higher frequencies. The attenuation for the least expensive coaxial cable (in table 11.1) may be regarded as typical for coaxial cables ([99] Fig. 2-15). The more expensive H100 cable has smaller attenuation (1/7 at 1GHz) to a higher price (4 times as high).

Table 11.2 shows some of the characteristics for a fibre available from one of the suppliers to the Norwegian market. The price is similar to that of the most expensive coaxial cables. More expensive optical fibres with attenuation down to 0.1dB/km are available. Low attenuation is important for reduction of power consumption. The receiver needs an amount of energy above a minimum threshold for sufficient data detection. $1\mu W$ is a typical minimum value for standard optical fibre sensors. The source has to contribute with an energy also compensating for the loss in connections and fibre. The loss in the connection between the light source and fibre is significant and often dominating for short fibres. A loss between 8dB and 25dB at this connection point is not unusual. Standard light sources for optical fibres typically transmit with an optical energy of at least $1000\mu W$.

Dispertion or smoothing of signal shapes results in limitations on how densely data bits can be transmitted on an optical fibre. One reason for dispersion is that the same part of a signal arrives at the destination at different times because of a difference in signal path. Dispersion due to different signal paths is reduced through narrowing the width of the fibre, to reduce the difference in signal path lengths. Another alternative is through "graded index", i.e. to increase the velocity of the signals travelling the longest path. This dispertion is given as a bandwidth-fibre length product. If the fibre length is reduced to $1/x$, the bandwidth may be increased x times. In the example in table 11.2, with a 1300nm light source the bandwidth is 250MHz for a

	Step-Index Multimode	Graded-Index Multimode	Single-Mode
Light Source:	LED or LASER	LED or LASER	LASER
Bandwidth:	Wide (up to 200 MHz · km)	Very wide (200 MHz to 3 GHz · km)	Extremely wide (3 GHz to 50 GHz · km)
Splicing:	Difficult	Difficult	Difficult
Typical Applications:	Computer data lines	Moderate-length telephone lines	Long telecommunication lines
Cost:	Least expensive	More expensive	Most expensive
Core diameter (μm):	50 to 125	50 to 125	2 to 8
Cladding diameter (μm):	125 to 440	125 to 440	15 to 60
Attenuation (dB/km):	10 to 50	7 to 15	0.2 to 2

Table 11.3: Overview of the main characteristics of the three most used classes of optical fibre.

	Price (USD):	Attenuation	V_{in}/V_{out}	Energy part at receiver
Coaxial cable:				
M17	133,-	238dB	$6.3 \cdot 10^{23}$	$1.6 \cdot 10^{-24}$
H100	480,-	32dB	$1.6 \cdot 10^{3}$	0.06%
Fibre:				
ELFA 55-955-25 : 850nm	530,-	0.7dB	1.17	85%
1300nm	(530,-)	-0.2dB	1.05	95%

Table 11.4: Coaxial cable and fibre compared for a 200m connection with a 1GHz signal.

2km long fibre and 1GHz for a 500m long fibre.

Another reason for dispertion is different signal velocity for different light frequencies. On optical fibres this is limited through choosing a light source with a more narrow frequency spectrum, like choosing a LASER instead of a LED. Table 11.3 ([93]) shows an overview of the typical characteristics of the most important three classes of fibres.

11.2.3 Conclusion

The choice of transmission medium and transmission methods between net nodes depends very much on the physical distance between the nodes to be connected. For a bandwidth of 1Gbps several twisted pairs in parallel would give an attractive solution for distances of about one or two meters. The latency will be low and the encoding and decoding circuitry needed simple. For larger distances at the same bandwidth, optical fibre is an attractive choice. For a few hundred meters the least expensive class of fibre, the step-indexed multimode fibre, can be chosen. For longer distance the fibre has to be replaced with a graded index multimode and for several kilometres a single-mode fibre has to be used.

For a general network system where most of the circuitry is integrated, a natural strategy would

be to integrate the required parts of the switch and bypass functions when not needed. Thus, the encoding and decoding circuits between serial and parallel transmission should be integrated but could be bypassed when the physical distance between net nodes are small. Based on such a strategy encoders and decoders for serial transmission would be included on all net node connections. As concluded elsewhere in this thesis, network latency has a minor influence on network performance compared to bandwidth and network interface latency. Thus, the latency added by using serial encoding and decoding circuitry where direct parallel transmission would be faster, is of minor importance. The choice between parallel twisted pair or serial fibre on short distances is more a question of what is practical and of low cost than of network performance.

Chapter 12

Flow control and input buffering

The subject of this chapter is the local flow control system used between the net node buffers. The function and path of the flow control signals are described. The chapter also contains tables of minimum and recommended buffer sizes.

"Flow control" is a system where control signals are transmitted upstream to start and stop the down stream data flow. This is used to prevent buffers from overflowing. Flow control can be used between buffers on different levels. Examples are: between net nodes, between Protocol Engines and between variables in processes. In this thesis the term "flow control" is used in the meaning flow control between net nodes, when nothing else is stated.

Flow control in SWIPP.

SWIPP uses flow control to prevent buffer overflow and resulting packet loss.

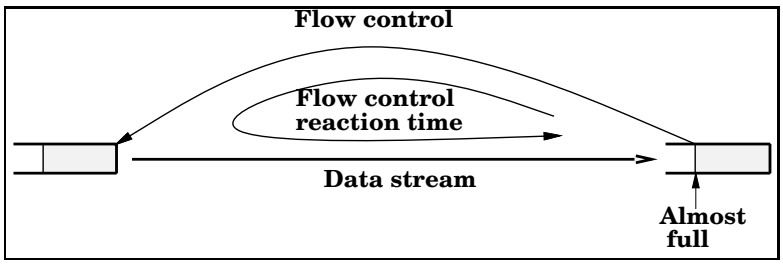


Figure 12.1: Flow control

The flow control signal transmitted upstream (see figure 12.1) tells whether the amount of data in the receiving buffer is more or less than an "almost full" value. Whenever the amount passes the "almost full" mark in the increasing direction, "stop transmission" signals are transmitted. "Start transmission" signals are transmitted when the amount of data in the buffer passes the "almost full" mark in the decreasing direction. The flow control signal is transmitted as the middle bit in all passing upstream control symbols. A change of flow control status is always immediately transmitted. If a control symbol is not passing by, a new control symbol will be created and inserted into the data stream.

Figure 12.1 is a sketch of a flow control system. A flow control signal is transmitted upstream to

the previous larger buffer.

> *The **flow control reaction time** is the time from a flow control signal leaves an Input Port FIFO until the packet stream into the same buffer has changed according to the transmitted flow control signal.*

All channels used in SWIPP are full duplex. Physically they consist of two wires or two buses transmitting in opposite directions. A small part of the bandwidth in both directions is occupied by flow control signals for data in the opposite direction. All control signals carry, beside their own value, one bit of flow control information. Control symbols may be found everywhere, both between and inside packets.

12.1 The SWIPP flow control explained by examples

12.1.1 The path of the flow control signal

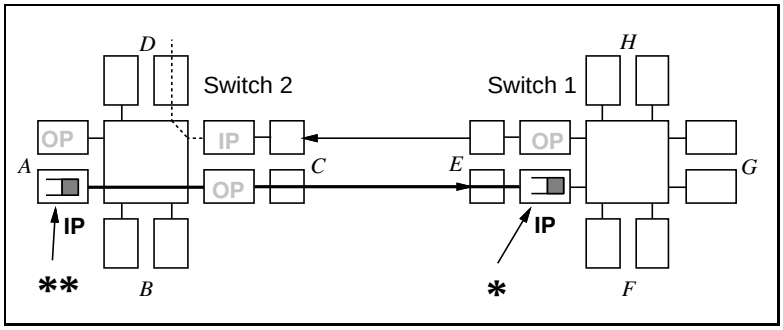


Figure 12.2: Path of the flow control signal: step 1

Figure 12.2 shows a segment of the SWIPP network. The section shown contains two switches. We will in our explanation start from the buffer marked by a single asterisk in figure 12.2. This is the Input Port buffer at channel E in switch 1. This buffer is fed with packet data from the buffer of channel A at switch 2. The feeding buffer is marked with two asterisks.

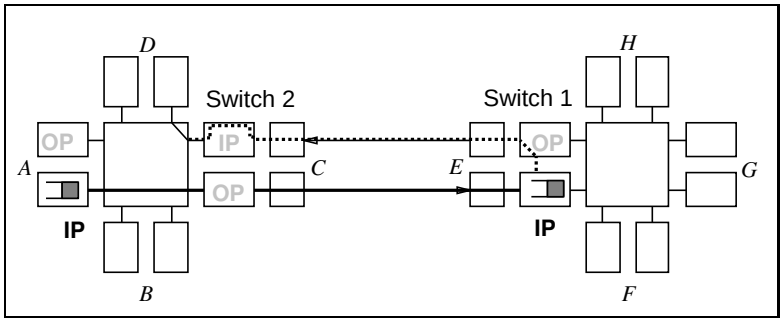


Figure 12.3: Path of the flow control signal: step 2

Locally the flow control signal of the A - E packet stream has a separate line from the E IP to the E OP. If this flow control signal changes, a control symbol will be created and inserted into

the data flow from E to C (upper line in the left direction). In the C IP, the flow control signal is transferred outside the buffer and bypasses data stored in the buffer of the IP.

The signal path is shown as a dashed line in figure 12.3.

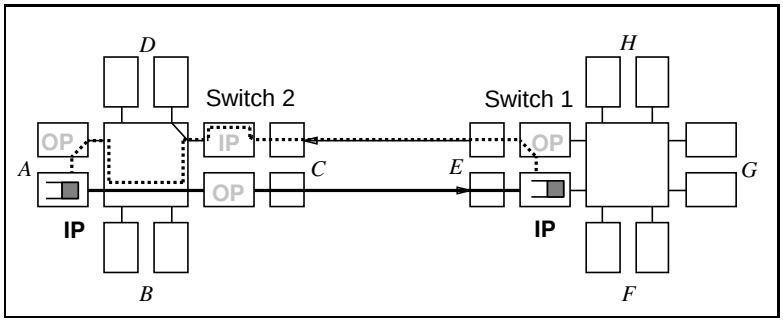


Figure 12.4: Path of the flow control signal: step 3

The E-C packet stream has another destination than the flow control signal. In our example the packet stream conveying the flow control signal is headed for channel D, while the flow control signal is going back to channel A. The flow control signal is separated from the conveying packet stream and follows the packet stream it belongs to through the CSU to channel A. This return path through the CSU to the left and the forward path from A to C are established simultaneously. The bus between Input Port A and the CSU is directed toward the CSU. Thus the flow control signal going the opposite way has to go via the Output Port of A .

The signal path is shown as a dashed line in figure 12.4.

12.1.2 Flow control signals generated by the Output Port

In the previous description we said that the E OP creates a flow control signal when the amount of data in the FIFO buffer of Input Port E passes the "almost full" mark. This flow control signal is inserted into a control symbol in the packet stream from E to C (fig. 12.5). If a control symbol is not passing by, a control symbol will be created. A new control symbol will require one position in the packet stream (fig. 12.6). Thus, the following data have to be delayed one period. A delay of one period requires one word of buffer space. Thus a small buffer is used for the insertion of such control symbols into the packet stream.

Idle symbols can be removed from the packet stream. Hence the occupied part of the OP buffer can be reduced. The used part of the buffer will increase if the number of control symbols which have to be inserted into the outgoing data stream is larger than the number of passing *idle* symbols. If the Output Port buffer is nearly filled, a small stop in the passing packet stream can be generated. This can be done by putting a short "stop" into the flow control signal passing through the peer Input Port.

The Output Port of C in figure 12.7 inserts flow control signals for the connection from E to C into the packet stream from A to E. If the Output Port Buffer of C is full, C has to signal this back to the Input Port feeding the C OP (in this case the A IP). This added signal path is marked by a small vertical dashed line from the C Output Port to the C Input Port in figure 12.7. The Input Port of C performs an OR function between the signal from the C Output Port and the signal from the E Output Port and sends the result of this logical OR to the A IP.

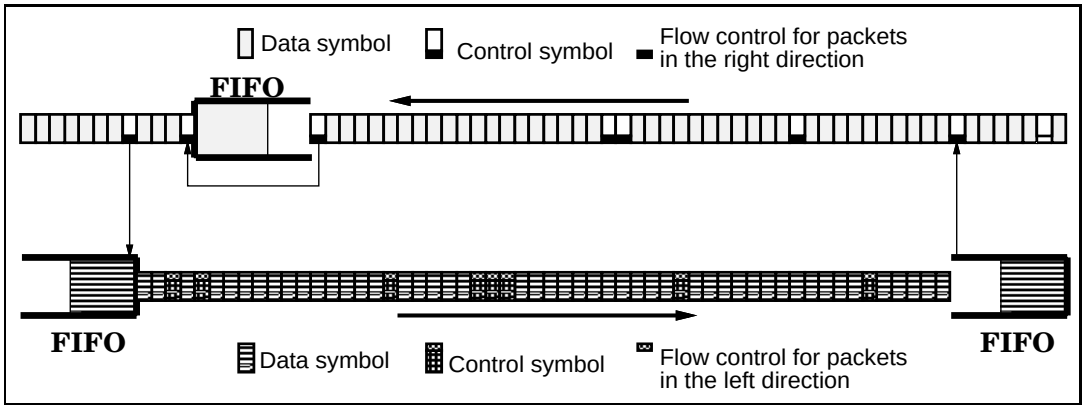


Figure 12.5: A flow control signal has to be transmitted from the lower right FIFO. By coincidence a control symbol is passing by in the upper, left directed, stream. The flow control signal is inserted into the passing control symbol.

The signal from the C Output Port to the C Input Port is active only for a number of periods corresponding to the size of its own buffer.

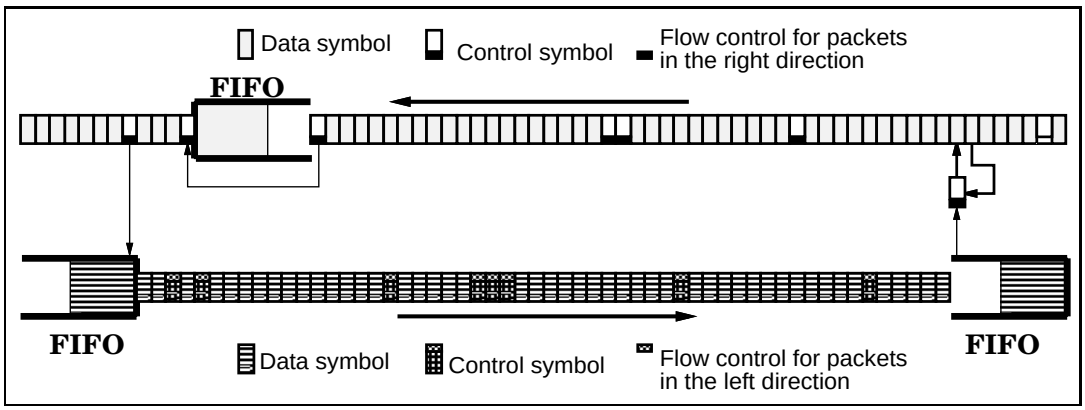


Figure 12.6: A flow control signal has to be transmitted from the FIFO in the lower right corner of the figure. Since no control symbols are passing by, a control symbol is created and inserted into the upper left directed packet stream. The flow control signal is inserted into this control symbol. To do this, a one-word delay buffer is inserted. The delay buffer between the data streams on the right side corresponds to the Output Port FIFO.

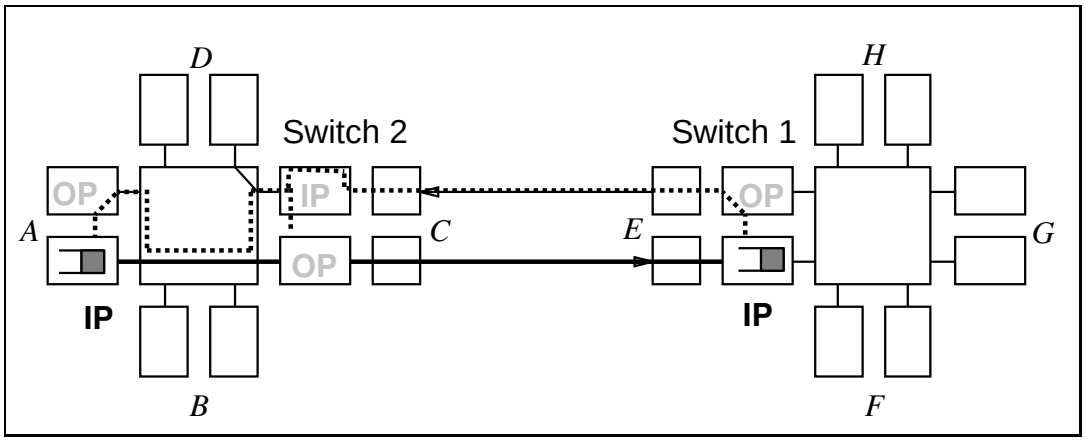


Figure 12.7: Flow control signals generated in the Output Port

Chapter 13

The arbitration logic of the CSU

More than one input channel may request the same output channel at the same time. The logic which selects between these input channels is called the arbiter. This chapter discusses the arbitration logic.

In SWIPP the arbitration logic is included in the CSU. This makes it possible to establish new connections faster than if the arbiter was external. The arbitration at each output channel is done by a private arbitration block, independently of the other output channels.

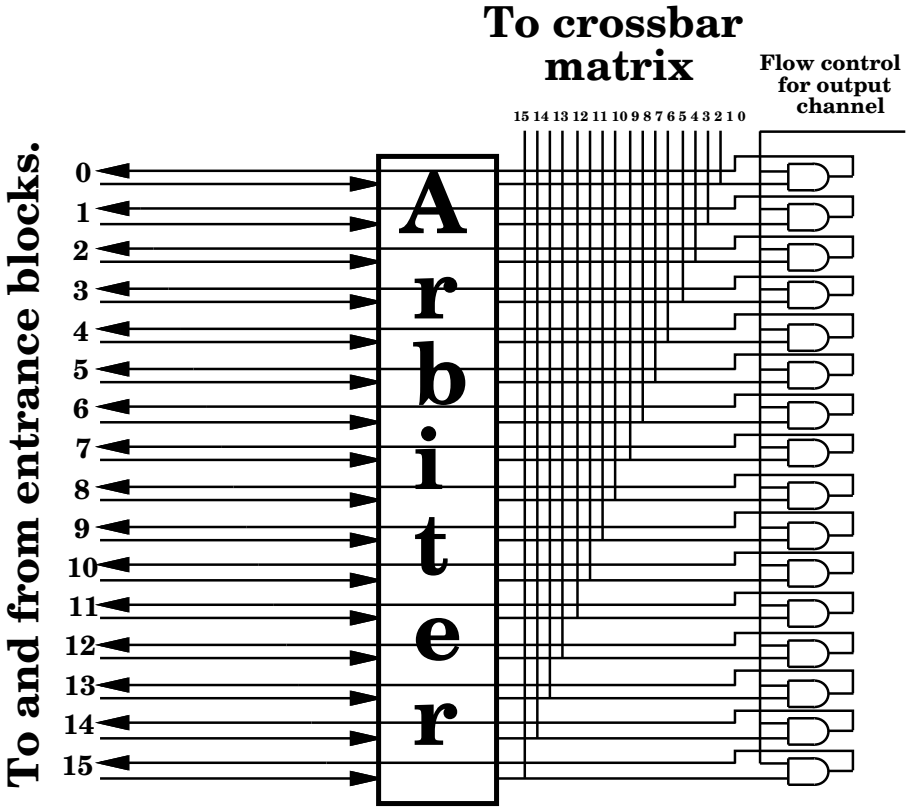


Figure 13.1: Arbitration block for one output channel of a switch with 16 input channels.

Figure 13.1 shows an arbitration unit for one output channel. The left side shows the 16 line pairs from the entrance blocks. The connection from each entrance block consists of the incoming **request** line and the outgoing **response** line. A high **request** line tells that the input channel requests for the output channel of the arbiter. A high value on a response line signals that the connection has been established and that the next buffer downstream is ready for data. As the figure shows, each outgoing connection is an AND function between the flow control signal received from the next buffer down stream and the result of the arbitration. The result from the arbitration block is used to control the crossbar matrix elements.

The logic task of the arbiter block is to put one or no output line high, while the remaining lines are kept low. One output is put high if one or more incoming request lines are high. No output lines are high if all incoming request lines are low.

13.1 Background: Arbitration algorithms in general

When two or more input channels request for the same output channel, arbitration is needed. There are different algorithms that can be used to choose which channel/packet should have highest priority and the sequence in which the remaining channels are allowed to send.

The first decision may be whether the system should be *preemptive* or *nonpreemptive* ([61] p. 108). In *preemptive* systems the packet under transmission can be interrupted to let another input channel transmit to the output channel. This is a reasonable alternative in systems where some packets are much more important than others. In *nonpreemptive* systems, the ongoing transmission of a packet will always be completed. SWIPP has chosen the *nonpreemptive* solution because this is the solution most easily implemented in hardware and because we expect the advantage of breaking packets to let other pass to be smaller than the disadvantage of restarting the broken packet.

Some algorithms only select between input channels requesting for the output channel of the arbiter, while others select between all input channels. An algorithm selecting between all channels may select an input channel that has not sent any request for the output channel this arbiter belongs to. If this happens, the algorithm used for the first choice may be used again or another algorithm can be selected. An algorithm which is easy to implement in hardware is the *cyclic second order* algorithm. In this algorithm the priority will decrease in a circle. The neighbour on one side has one step higher priority while the neighbour on the other side has one step lower priority except where the circle has been broken.

13.1.1 The algorithms

There is a large number of algorithms for different needs. For some algorithms the queue order is fixed from the time the queue members queue up for access to the output channel. These are algorithms like **FCFS: First-Come-First-Served** and **LCFS: Last-Come-First-Served**. Some algorithms are functions of the packet lengths like: **SJF: Shortest-Job-First** and **LJF: Longest-Job-First**. Priority-class algorithms separate packets into priority classes. The classes may be identified by values carried by the packets or by the input channels. We also have the **Random** algorithm.

The packet length algorithms

Some algorithms give priority to packets depending on their length. Two such algorithms are named **SJF: Shortest Job First** and **LJF: Longest Job First**. We have chosen not to have any packet length information in the packet header. This choice makes it possible to start transmission of a packet during the creation time of the packet, without knowing how long the packet is going to be. This is an advantage in some cases. Since we have no packet length information in advance, the packet length algorithms can not be used.

The algorithms not depending on packet length.

According to L. Kleinrock's queuing theory [61] page 113, when the arbitration is independent of packet lengths, the average number of packets in the system and the average waiting time are both independent of the order of service. Thus, we can not find any "clever" algorithm that reduces the average waiting time or packet number.

The fixed priority algorithm.

In this algorithm each input channel has a fixed priority rank. Thus one input channel will always have the highest priority, while another always has the lowest priority. This may give an unfair throughput and differences in average waiting time between the input channels. The unfairness of this arbitration algorithm will be discussed in subsection 13.1.2.

The advantage of this algorithm is its simple implementation. It is the algorithm requiring the smallest number of transistors. The implementation can be done as two daisy-chains.

The LSLP: Last-Served-Lowest-Priority algorithm.

In this algorithm, (also known as Round-Robin) the input channel that was last served will be given the lowest priority in the next arbitration. As a result, the priority levels will be shifted around so that all input channels have an equal opportunity to transmit. The implementation of this algorithm is more complex than the previous one. It can be implemented as a daisy-chain with two memory cells for each input channel. The simplest and straight-forward implementations of this algorithm are noise sensitive. The logic will not recover from wrong states due to erroneously generated '0' and '1' bits in the control logic. Logic detecting and handling this should be added.

The Random algorithm.

In this algorithm the input channel with the highest priority is selected at random. This algorithm is often implemented with a fixed priority sequence following a passive first choice channel. In this case the Random algorithm is more fair than the fixed priority algorithm, but less fair than the Last-Served-Lowest-Priority algorithm. An input channel following a sequence of passive channels will have much larger possibility to transmit than one following an active channel.

The FCFS: First-Come-First-Served algorithm.

This algorithm is the most fair one of all. The input channels are served in the sequence they request the output channel. The disadvantage is that the algorithm is more complex to implement. The arbiter has to store both the identity of the requesting input channels and the order in which the requests are presented.

13.1.2 The performance of the fixed priority algorithm

The fixed priority algorithm is the algorithm requiring the smallest number of transistors. Therefore that algorithm is an interesting choice. The disadvantage is that it is unfair. It is important to know how large difference this unfairness makes between the different channels. We will in the following study some load levels and look for differences in throughput and average waiting time depending on channel priority.

All simulation results are for a 16×16 switch. The input load is the percent of the time the input channel is fed with data. The throughput is the fraction of the time for which data are read out of the input channels. If the throughput is less than the input load the channel is in saturation. In saturation the amount of data in the (here: infinite) input buffers will be increasing.[1] In saturation the average waiting time will also be proportional to the simulation time. All input channels select each output channel with equal probability. The average waiting time is given relative to the average service time (i.e. packet bandwidth time t_{pk-bw}). The x-axis is the different input channels with decreasing priority for increasing x.

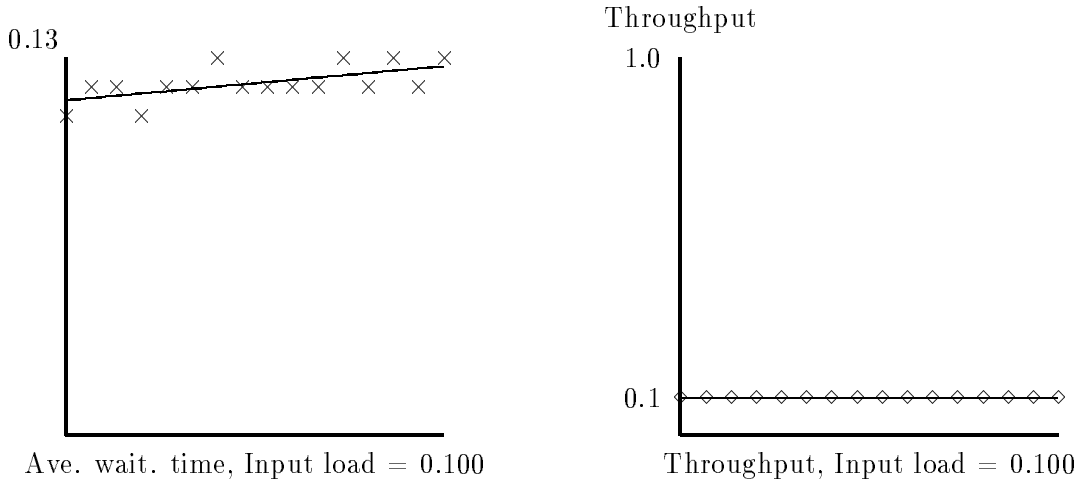


Figure 13.2: Simulation of average waiting time and throughput for a 16×16 switch with an input load of 10%. Along the x-axis we find the different input channels, priority decreases with increasing x. The lines are calculated linear regress lines found from the simulated data.

Figure 13.2 shows the simulation result when all input channels are loaded 10% of the time. We see from the figure that the throughput is approximately 10% for all channels. Hence the network can handle this input load. We also see that the difference in average waiting time is small. We

[1] This increase is the product of the simulation time and the difference between the throughput and the input load.

read from the figure that the normalised average waiting time is approximately 0.13. This means that if a packet needs a time t_{pk-bw} to pass without waiting time, it will in this case wait a time $0.13t_{pk-bw}$. This gives that the packet will spend a total time of $1.13t_{pk-bw}$ in the switch.

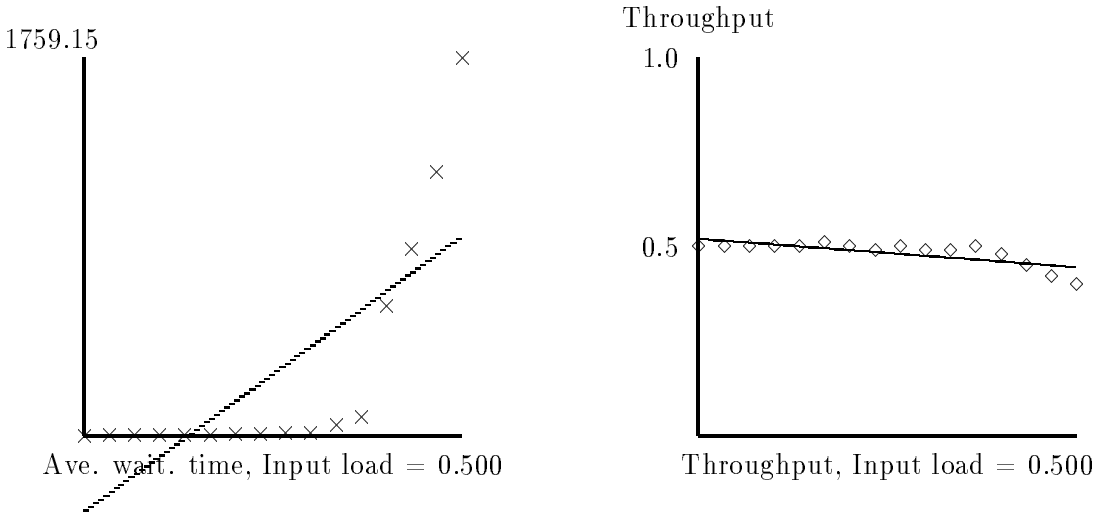


Figure 13.3: Simulation of average waiting time and throughput for a 16×16 *switch with an input load of 50% . The lines are calculated linear regress lines found from the simulated data. From the figures it looks like the waiting time is linear below the saturation point (previous figure) and above the saturation point (next figure). Around the saturation point (this figure) it looks like the waiting time is exponential.*

Figure 13.3 shows the simulation result when all input channels are loaded 50% of the time. For an input buffered switch with a fair arbitration algorithm the saturation values would be between 52% and 58%[2]. We see that all input channels except the four with the lowest priority get rid of their input load. The channels with the lowest priority have a lower throughput and much longer waiting time than the others.

Figure 13.4 shows the simulation results where all input channels are loaded 80% of the time. In this situation no input channels get rid of their input load. Thus the amount of data in the input buffers is increasing.

We come to the following conclusions about the difference between fair and unfair algorithms when all input channels select between all output channels with equal probability: The difference is negligible with 10% input load and significant with 80 % input load. If the distribution of output channels is not even i.e. some output channels are preferred before others, the difference will start at a smaller input load. In the extreme case where all input channels request for the same output channel only the two input channels with highest priority will have access to the output channel.

[2]This author's simulations

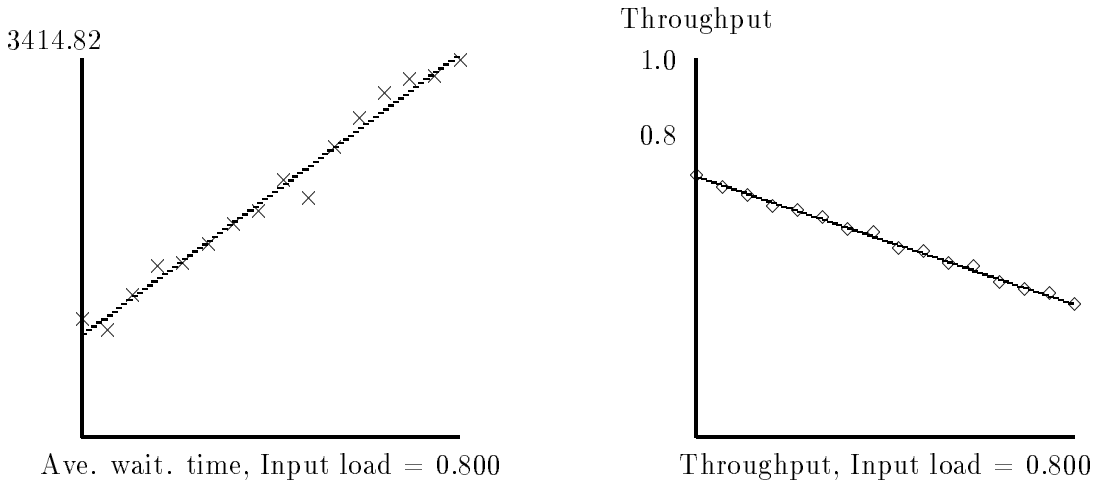


Figure 13.4: Simulation of average waiting time and throughput for a 16×16 *switch with an input load of 80% .*

13.2 Logical implementation of the arbitration blocks

This subsection gives some proposals for implementation of arbitration schemes discussed earlier in this chapter.

13.2.1 Unfair arbitration

Arbitration logic may be implemented as a daisy-chain with one bit of logic for each input channel. The signal going down the chain is 'output idle'. An 'output idle' indicates that no input channels until that point in the chain have been requesting for the output channel. A passive input channel will pass the daisy-chain signal unchanged. A requesting input channel receiving an 'output idle' signal can start transmission to the output channel. It will forward an 'output occupied' signal down the chain.

With this implementation the input channel closest to the beginning of the chain has the highest priority while the priority decreases down the chain.

A solution with one daisy-chain would be sufficient for *preemptive* arbitration, where a channel can break a transmission from an input channel with lower priority (longer down the daisy-chain).

In *nonpreemptive* systems, the ongoing transmission will always be allowed to finish. For such systems we will need another daisy-chain going in the other direction. This daisy-chain will tell whether any input channel with lower priority has already started transmission. In this system an input channel can only start transmission if it receives 'output idle' signals from both chains.

The arbitration decision is made by an array of equal cells. Each cell serves one input channel.

In the chain drawn in figure 13.5[3] highest priority is given to the leftmost bit with decreasing priority in the right direction.

[3]The implementation schemes given on this and the following pages have been found by the author. They are believed to be straightforward and thus not unique. A limited search for references, however, has given no hits.

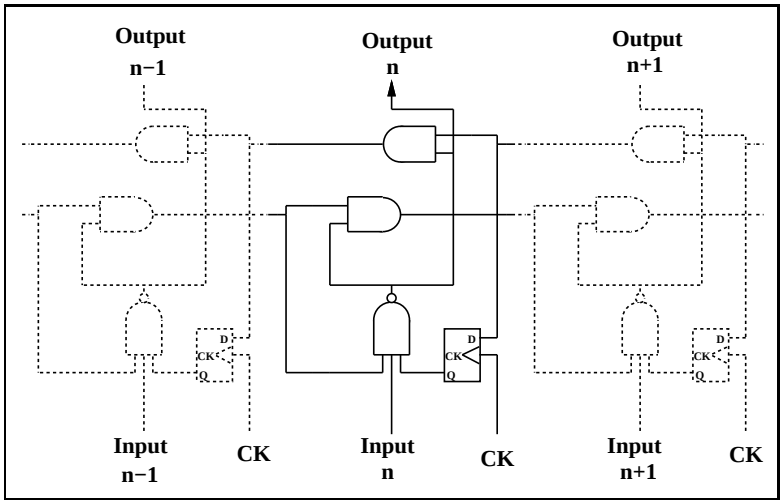


Figure 13.5: Cell in unfair arbitration block.

We conclude that the priority ranking will be:
1. connections with higher or lower priority which are already established, (left and right directed chains in the figures) and
2. connections with the highest priority (right directed chain in the figures).

To achieve knowledge only about established connections and not new requests, signals from cells with lower priority are delayed by one clock cycle.

In practical implementations we can not make AND ports but have to make NAND ports and NOR ports. For this reason we show how figure 13.5 can be changed into figure 13.6. In this way every second cell will have the same layout. Figure D.6 and figure D.8 in Appendix D show a scheme for an arbiter with a higher clock rate.

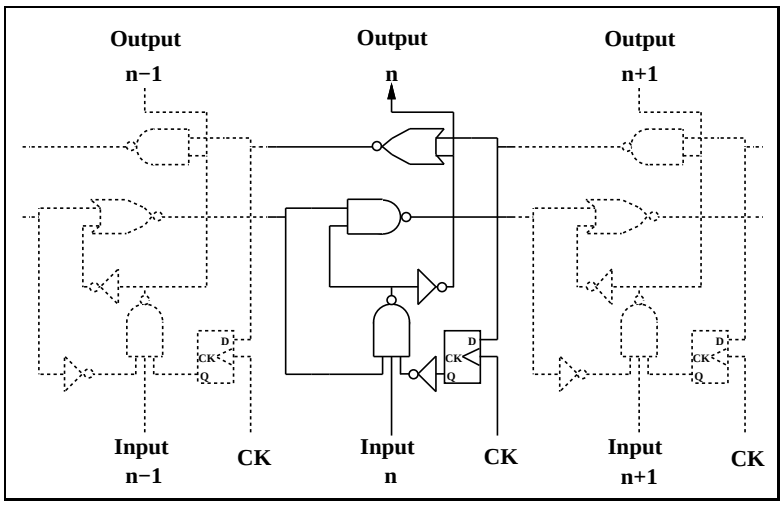


Figure 13.6: A solution with NAND and NOR instead of AND.

Figure 13.7 shows an arbiter block for 4 input channels. All input channels are passive. A high

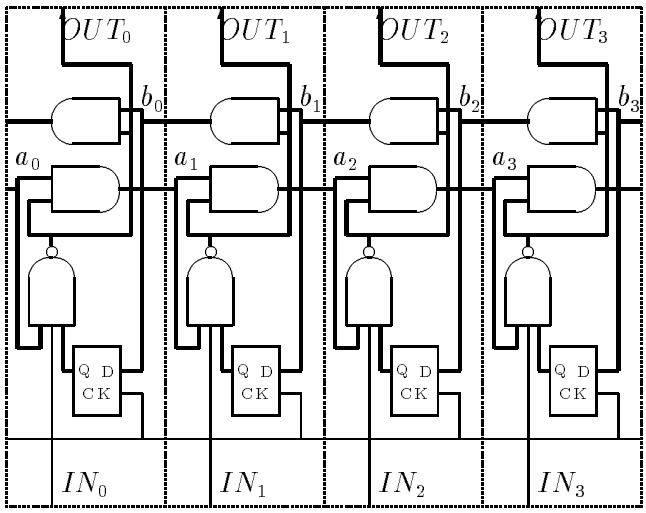


Figure 13.7: 4 bits of arbitration block: idle

level is indicated by thicker lines than for the low level. Notice that arbitration output signals (top signals) have inverted logic.

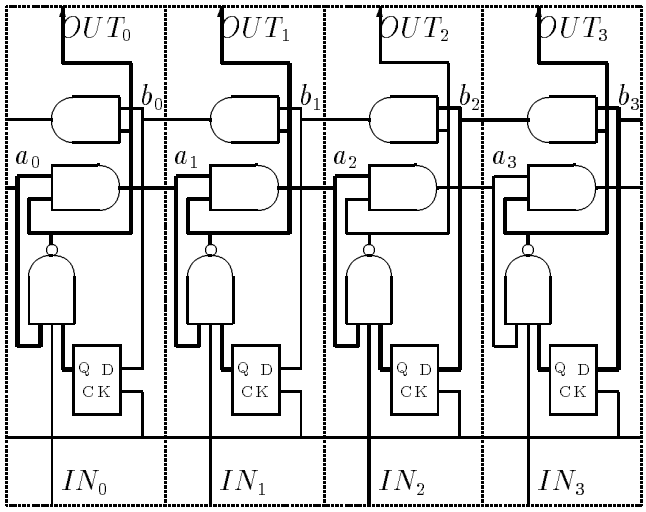


Figure 13.8: 4 bits of arbiter block: channel 2 requests for output channel.

Arbiter function illustrated by an example

Figure 13.8 shows an arbitration block where input channel 2 is requesting for the output channel. We see that the daisy-chain signal "output idle" is broken in both directions. Both input channel 0 and 1 may take over the output channel from input channel 2. This has to be done before the daisy chain values are latched. In the next period channel 2 will maintain the connection to the output channel independently of requests from the other input channels.

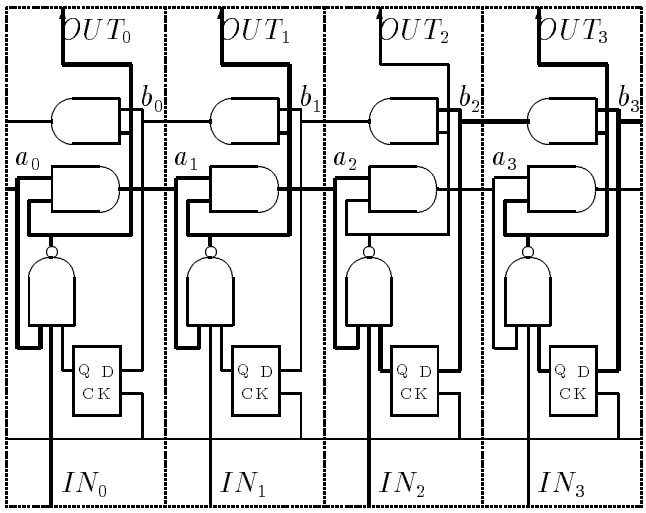


Figure 13.9: 4 bits of arbiter block: channel 2 sends to the output while channel 0 applies.

Figure 13.9 shows the step after the one described in figure 13.8. Here channel 2 has already established the connection and channel 0 is requesting for the output channel. Channel 2 requested for the output at least 1 clock period before channel 0.

13.2.2 Fair arbiter cell

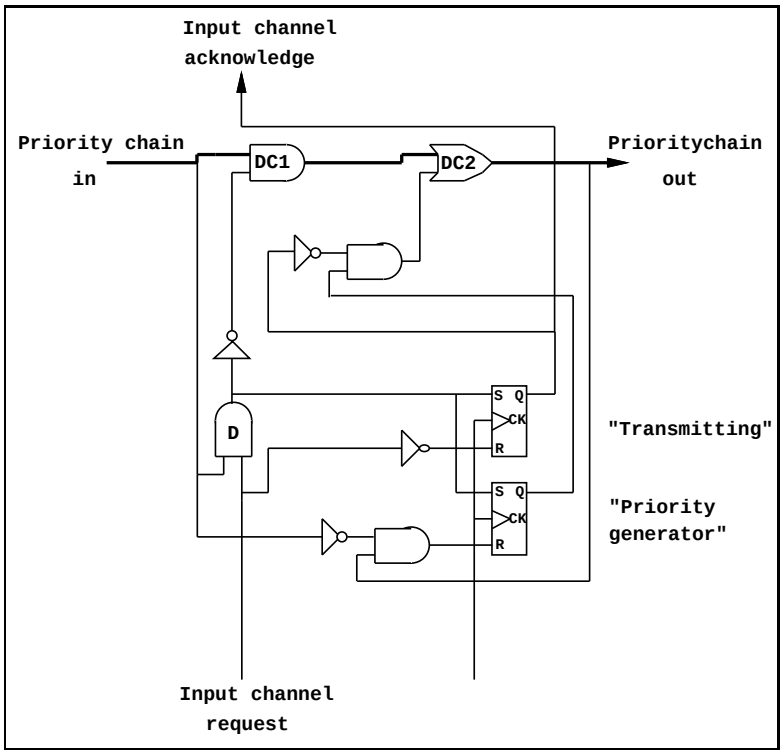


Figure 13.10: Cell in fair arbiter block. This arbiter has a Last-Served-Lowest-Priority algorithm. After this cell has transmitted, it is given the lowest priority, the next cell to the right has the highest priority and the next cell to the left has the second lowest priority.

Figure 13.10 shows a cell in a fair arbiter block. When we compare with the cell drawn in figure 13.5 we find that the fair cell has only one daisy-chain signalling in the right direction. We also find that the circle is closed. We also notice that the new cell has two memory bits.

The priority chain is drawn with a thicker line in the upper part of the figure. A requesting input channel can transmit if it receives a positive signal from the chain. It will then turn off the priority transmission to the next stage (with gate **DC1**). After it has finished transmission it will generate a positive signal ("output idle") for the daisy-chain through gate **DC2**. This is done until the circle is broken, i.e. until another input channel is requesting for the same output channel. When this happens, the cell will stop generating the priority signal, leaving this "token" for the new requesting input channel.

The cell has two registers labelled **transmitting** and **priority generator**. They are one-bit memory cells with clocked set/reset. **Transmitting** is set when the input channel is requesting and the priority chain inputs are high. It is reset when the input channel is cancelling its request.

Like the **transmitting** register the **priority generator** register is set when the input channel can start transmission. As indicated above, the register output is used to generate a priority signal for the priority chain. This will be generated when the **priority generator** is active, while the **transmitting** register has been turned off. When there is a difference between the incoming and outgoing priority chains, i.e. some other channel has taken the control of the daisy chain, the **priority generator** will be reset.

13.3 Arbiter in separate clock domain

Clock rate (and bandwidth) is adjusted to the longest propagation time between latches. Thus it is important to minimise the propagation time. In the arbiter implementations described in this chapter, a signal has to propagate all the way down the daisy-chain during one clock period. To avoid the clock rate from being influenced by such a long delay we have several choices. One solution is to let the arbiters run on a slower clock generated from the master clock. Other possibilities are to use bypass logic where the chain signal bypasses groups of passive input channels or to use a solution like the one in figure D.6 and figure D.8.

The best solution is probably a combination.

13.4 Conclusion and the choice for SWIPP

We conclude from the simulation figures presented in this chapter that fair arbitration is important for good efficiency. Unfair arbitration adds an unstability factor making traffic analysis and thus sizing of buffers and capacity more difficult. The fair arbitration scheme **LSLP: Last-Served-Lowest-Priority** presented should be a good solution for most cases. The unfair arbitration may be preferred when the area is very expensive, like in ECL. SWIPP switches has been designed both with LSLP and fixed, unfair arbitration.

It should be noted that unfair arbitration may also in other cases be an advantage. Connections from switches may be connected to the input channels with the highest priority while connections from Protocol Engines are connected to lower priority channels. Thus emptying the network is given priority over filling it.

Chapter 14

Error handling

In this chapter the subject is faults influencing network performance and network functions. We will look at which errors may occur, their possible damage, which precautions that can be taken to avoid them, and how the damage can be reduced if they should occur.

14.0.1 Types of errors.

Fault situations may occur from a number of different sources. We expect the main sources to be:
a) Bit errors from electrical noise, light dispersion, etc.
b) Fatal combinations are combinations of events expected to be rare and intended to be handled specifically when they occur, and
c) Design errors are software or hardware errors due to missing design overview.

a) Bit errors.

Some of the numeral sources for noise are spikes on the feeding power lines and noise due to capacitive coupling to neighbour lines. In optical fibres bit errors from dispersion due to different frequencies or different travel distance may occur. The error probability per bit increases between linearly and exponentially with the clock frequency. Thus, a maximum average error probability per bit may be achieved by reducing the bit rate. The acceptable error probability per bit depends on the system and its purpose. If links are regarded as the most expensive part, high bandwidth may be stressed and error correction often used. In this case a somewhat higher error level would be accepted. If, on the other hand, simple communication protocols are stressed, error correction should be rare and the acceptable error rate low.

Increased knowledge and improved technology have given lower error rate for higher frequencies. This influences topology, architecture, technology and the design of the individual electrical circuits. On the topology level, the error rate may be reduced by having only one transmitter and one receiver per link. Link segments operating on several hundred bits per second have been reported operating for months without errors ([90]).

b) Fatal combinations.

Two or more packets may by chance require buffer or channel resources in such a way that further packet propagation is impossible (Dead-lock). One way to avoid such a situation may be to put limitations on how and when packets are routed. If these limitations reduce the average transport efficiency too much, and the expected "unlucky" combinations are rare, it may be a better total solution to make sure that the problematic situations are handled separately.[1] Thus, single packets are rejected instead of the more global actions like rejecting all packets and resetting the total system. Clearly there has to be a trade-off between the probability of the damaging situations and the global consequences of how the problems are handled.

c) Design errors.

In large designs both of ASICs, software, and complex network topologies, it is almost impossible to get a total overview of all system situations and how they should be handled. Thus, there may be a possibility that unexpected situations arise, giving an erroneous behaviour as an outcome. Typically many design errors will be detected during prototype testing and result in modifications of the first designs. In complex systems some errors may be so rare that they are difficult to track or they may not occur during the initial test periods. Some precautions may be taken against such unpredictable errors to limit the possible damage they can make.

14.0.2 Results of errors.

The most common results of the errors discussed in the previous text are:
- Packets arriving at the correct destination but with one or more bits erupted,
- Packets routed to wrong destinations,
- Packets lost in the network, and
- Packets blocked due to circular requirement of the same resources (Dead-lock).

The first three of the listed results have in common that they will influence only the transmitter and receiver. The malfunction described in the last point may grow and influence the total network. Thus, it is especially important to avoid or reduce the probability for this error situation.

14.0.3 SWIPP: Avoiding errors and reducing their effects when they occur.

In networks, in general, error detection and error correction may contribute considerably to the latency. In SWIPP the error rates have to be so small that error checking can almost be avoided in switches and simplified at the receiving end. This must be remembered when clock rate and bandwidth are being increased.

The error handling in the SWIPP network may be divided into three points:
- Precautions taken at the transmitting end,
- Detection and damage reduction in the switches, and
- Detection and correction at the receiving end, if necessary with "help" from the transmitting

[1] I.e. the SWIPP timeout system where single packets are dropped when no propagation has taken place for 50ms.

end.

Precautions taken at the transmitting end.

The most serious malfunction, dead-lock, can be avoided through the choice of routing algorithm. Independently of the physical topology the routing algorithm may regard the topology logically as a tree topology or as an orthogonal two-dimensional mesh topology. These routing algorithms may be made dead-lock-free under the condition that noise does not alter address fields. An example is given in figure 14.1 [72].

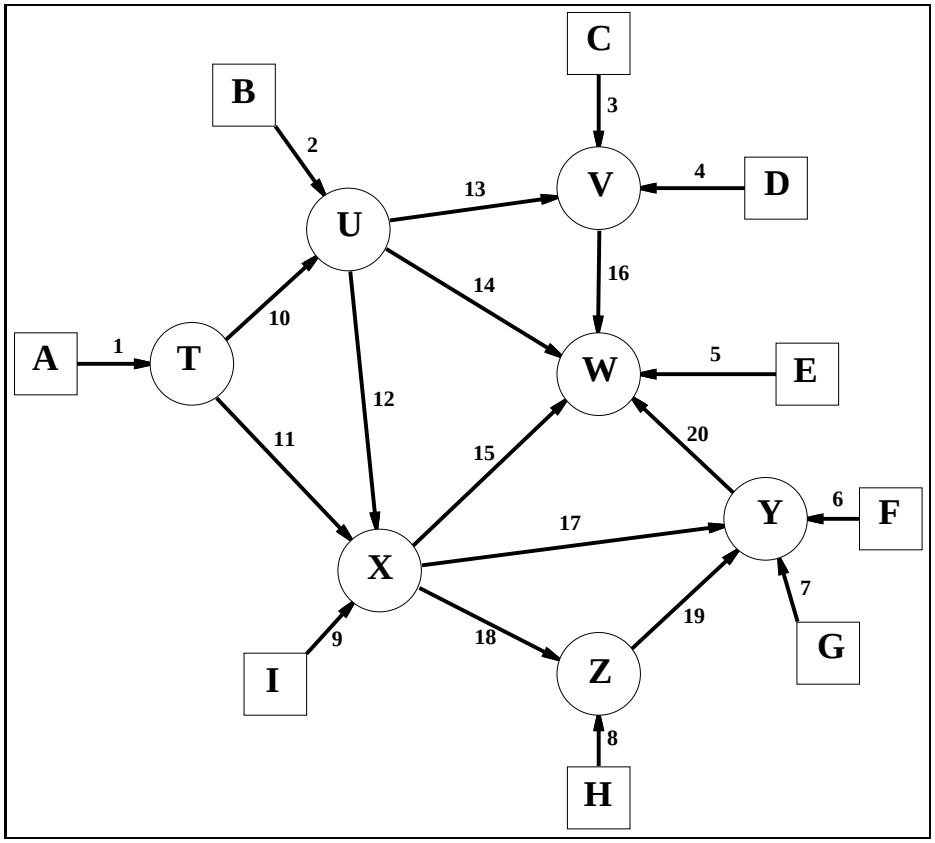


Figure 14.1: Logical directions used by a dead-lock-free routing algorithm. All links are full duplex connections. The distributed address procedures agree upon a logical "direction" for all links. The arrow-head indicates a logical "UP"-direction. A packet routing path is made so that an "UP" direction will never follow a "DOWN" direction. Thus the routing path consists of a sequence of "UP" directions followed by a sequence of "DOWN" directions. As long as this strategy is followed, the system will be dead-lock-free. Note that electrical noise may alter link addresses so that the strategy is broken. This routing algorithm gives some reduction of network utilisation compared to an unrestricted routing algorithm.

By adding error detection bits calculated from the transmitted data, i.e. parity bits or cyclic-

redundancy-code, error detection at the receiving end is simplified. By adding more redundant bits, error correction can be performed by the receiver without retransmission. Adding redundant bits makes it possible to reconstruct bits erupted by noise.

Another precaution that may be taken by the transmitter is keeping a copy of the transmitted packet until a positive acknowledgement signal has been received from the destination. If a positive acknowledgement signal is not received within a certain time the packet will be retransmitted. Thus, packets rejected by the network may have a new chance to pass through.

Error detection and damage reduction in the switches.

The switches perform no cyclic redundancy check or parity check on passing packets. Cyclic redundancy checking would have required that a larger part of a packet was collected and checked before it was forwarded to the next net node. This would have increased the time to forward a packet from source to destination. The error rate is expected to be so low that performing data-check only at the end nodes can be defended. The data parts of the packets are not changed by the switches, and the end nodes may use CRC on these parts to confirm correct transmission.

The packet header is changed in every Input Port. Thus, checking and generation of new Cyclic Redundancy Code (CRC) would have been necessary in every Input Port. Compared to the present solution of SWIPP, this would make the packet header larger and more than double the time before a packet could be forwarded. Since the error rate is expected to be small, this author consider the advantage of CRC not to defend its cost.

The switches interact only when a packet does not seem to propagate. When a packet has not propagated within a certain period (50ms) it will be dismissed. This will open dead-locks and reduce the load on very heavily loaded connections. This should be rare since software protocols are expected to avoid dead-lock situations and discover heavy loading before switches start to reject packets.

Detection and correction of errors at the receiving end.

Depending on the quality of service requested for the connection, different methods of error detection and error corrections may be performed.

At the receiving net node both correct host and task address are confirmed. This is done by analysing address patterns and processing information in the packet head and comparing it with locally stored information. The receiver may ignore a wrongly routed packet and leave it to the source or destination to discover the packet loss. A higher quality of service would be to forward the packet to the correct destination or to inform the packet source.

A packet received at the correct destination may contain erupted bits in packet head or packet body. If the application accepts small bit errors, like when the packet contains sound or picture, the contents may be used without corrections. If the packet contents have to be accurate, data must be corrected. One way is to add additional redundant bits to the packet contents. These additional bits contain information about the ordinary data bits in such a way that it is possible to reconstruct the original data. The added bits have their limitation in that they can correct single bits or smaller burst of bits. If the number of erupted bits are too many the additional bits fail to correct them.

If no error correction bits are included, or if error correction bits *are* included but the errors too

many, retransmission from the source may be requested. This will take some time since a request message has to be transmitted to the source and the packet again forwarded from the source to the destination.

PART 4

HIGH LEVEL SIMULATION

Chapter 15

VHDL simulation of a switch circuit and a small network

The switch circuit has been described and simulated in the hardware description language VHDL. This chapter gives some examples from simulations of one switch and of a network consisting of four switches. The VHDL representation has been used to verify the flow control signalling and the signalling between the switch modules.

During the development of the switch architecture the architecture has been described in numerous versions in the *Simula* and *C* languages. In the beginning of this project, the simulations of the switch showed behaviour unexpected by the author. As a result of this observation new representations of the switch architecture were designed. The new representations belonged to one of two types: one *network traffic dependent*, written to give the switch architecture reaction to different load patterns and load levels, and one *hardware behaviour dependent*, used to simulate the implementation of the hardware. The simulation results of the representations of the first type, together with knowledge about applications, influence our implementation of the hardware and thus the simulation code for the latter type. Except for this, the two classes cover different fields. Their simulation results are not compared. The results from the load simulations of the first class of representation are presented in chapter *22, Traffic modelling of an input buffered switch*, and will not be discussed further in this chapter.

15.0.4 The VHSIC Hardware Description Language (VHDL)

New languages dedicated for description of digital electronic and digital systems have been developed. One such language is **VHDL**. VHDL is a result of a research program started by the US Department of Defence in 1980 in *Very High Speed Integrated Circuits (VHSIC)*. It was clear during the research program that a standardised language for description of structure and function of integrated circuits was needed. Thus, the *VHSIC Hardware Description Language (VHDL)* was developed. Later VHDL has been accepted by *Institute of Electrical and Electronic Engineers (IEEE)* as standard **IEEE 1076-1987 VHDL**.

VHDL offers the possibility to describe hardware in different styles separately or mixed. The hardware may be described structurally, behaviourally, functionally, combinatorial and/or as a data flow description. Through a structural description, the hardware may be described as units and sub-units. The behavioural description may be given at a high level with one sentence representing an adder function or with a number of sentences describing the specific logical ports

of an adder.

15.0.5 The Synopsys Graphical Environment (SGE) for VHDL.

The software developer **Synopsys** offers a graphical interface to VHDL named SGE: Synopsys Graphical Environment. This tool has been used for the last version of the switch architecture. All schemes and simulation results presented in this chapter are generated with this tool.

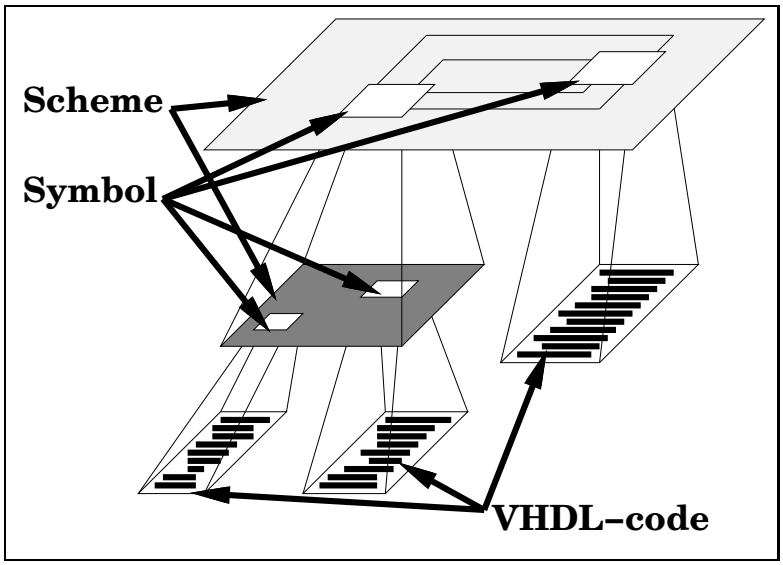


Figure 15.1: The SGE interface to VHDL.

Figure 15.1 illustrates the SGE hierarchy. The top level scheme consists of symbols and wires and shows how the symbols are connected. The symbols represent either VHDL code or new schemes at the first level below. The leaf cells at the bottom of all branches are VHDL code.

For simulation, VHDL code is generated for all schemes, so that the entire architecture is described in VHDL.

15.0.6 The SWIPP switch described in VHDL.

The VHDL code of the SWIPP switch makes it possible to simulate more switches together and still have a detailed description of the logical implementation. This is important for verification of flow control protocols and other signals transmitted between switches. It is important that the description is detailed so that the VHDL code represents the *real* implementation and not a simplified higher description.

The VHDL code is quite detailed. Simple register cells are represented as variables and sometimes gates. Combinatorial logic is represented either as Boolean equations or as gates. The level of detail should make it easy to compare layout with VHDL code. The only part not described at a low level is the elastic FIFO buffer of the Input Ports. This is due to its asynchronous behaviour which is a little more difficult to describe in VHDL.

15.1 The schematic representation of an 8×8 switch.

An 8×8 SWIPP switch has been designed with SGE. The 8×8 size was chosen instead of the full scale 16×16 due to some limitations in SGE. All simulations, both for the single switch and for the 4-switch network are with this 8×8 switch.

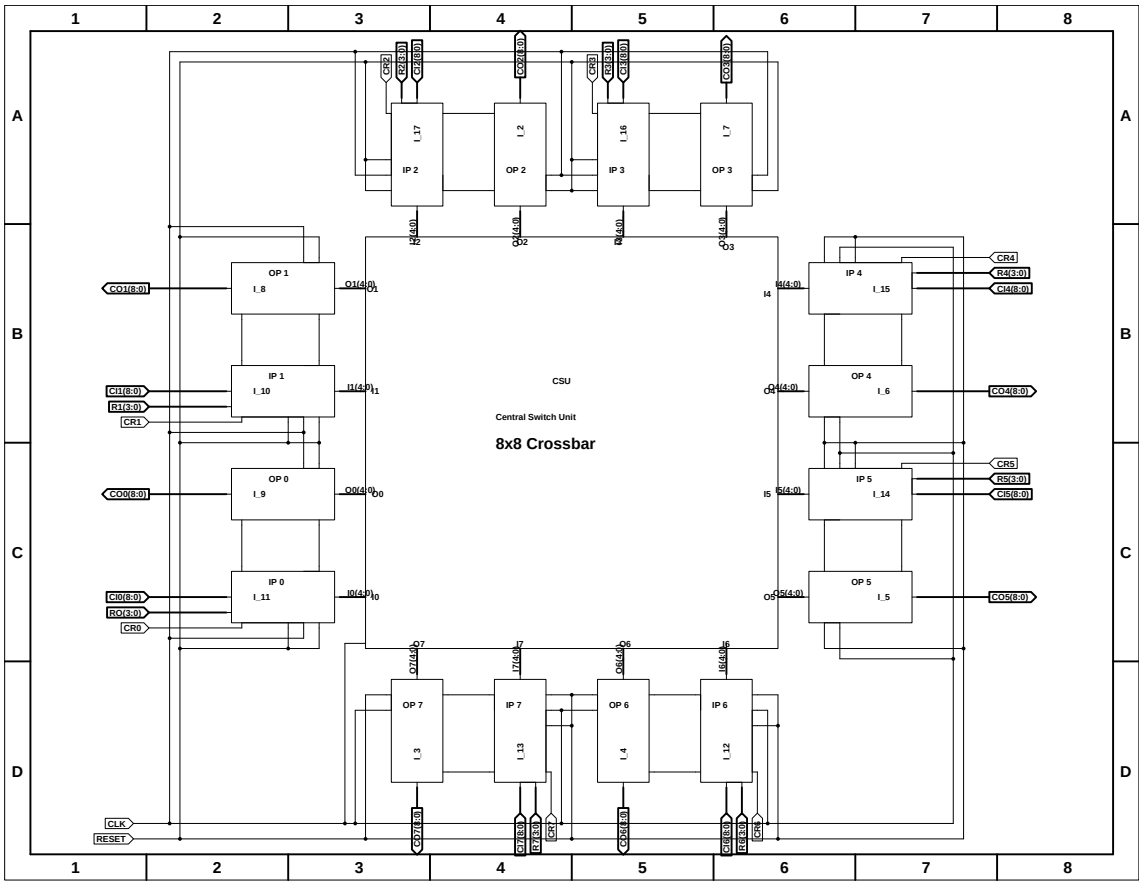


Figure 15.2: The SGE representation of an 8×8 switch.

Figure 15.2 shows the SGE representation of the 8×8 switch. We have the 8-channel CSU in the middle and 8 pairs of Input and Output Ports outside this.

The external connections of the VHDL switch are:
— the `clk` and `reset` signals routed to all symbols,
— the 16 `CIj` and 16 `Rj` buses routed one to each of the Input Ports,
— the 16 `CRj` signals also routed one to each of the Input Ports, and
— the 16 `COj` buses one from each of the Output Ports.
`CIj` is the input channel of Input Port *j*, while `CRj` is the remote clock of the transmitting end of the channel. `Rj` is the hard wired number of each channel i.e. the contents of `Rj` is *j*. `COj` is the output channel of Output Port *j*.

15.2 Simulation results for an 8×8 channel switch.

In the following, we will look at a packet passing through a switch circuit. We will monitor the

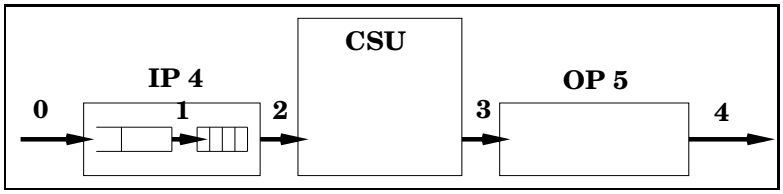


Figure 15.3: Some cross sections of a packet passage through a switch.

packet at 5 different points as given in figure 15.3. Point **0** is the channel 4 input. **1** is inside the Input Port between the FIFO and the pipeline used for decoding of the packet header. Point **2** is between the Input Port and the CSU. The bus between the CSU and the Output Port is point 3. The last point, point **4**, is after the Output Port.

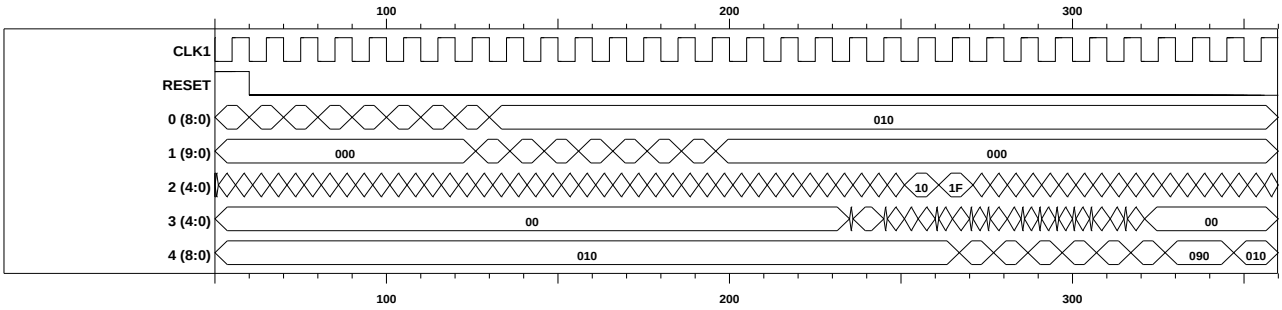


Figure 15.4: The simulated curves at the cross-sections given in the previous figure.

Figure 15.4 shows the simulated curves. The upper curve is the clock signal. Below is the reset signal with the waves for the cross-section points below in the order **0** to **4**.

The beginning of the arriving packet enters the Input Port at the $65ns$ point of time in figure 15.4. The packet consists of 7 symbols. With a clock rate of 100 MHz, the packet needs $70ns$ to pass a cross-section of the network.

In this simulation, each part of the packet needs $60ns$ to pass through the elastic input buffer. Most of this time is used for stable transfer of data between clock regions. A reduction in this time increases the probability for metastable states.

Each part of the packet needs $90ns$ to pass through the 9 steps of the pipeline between **1** and **2**. It is possible that a redesign would have reduced the pipeline with one or two steps. A reduction by more than two steps will require reduced clock speed. The product of the number of steps and the clock period will not change much and there will be a lower limit on the propagation time through the pipeline. When the packet head reaches the head of the pipeline a request for output channel is transmitted to the CSU. $30ns$ later a positive flow control signal is received from the CSU (if the output is ready) and packet transmission can start. One clock period is needed to forward a symbol through the CSU between point **2** and point **3**. One last clock period is required to forward the packet symbol through the Output Port.

Point	Time in figure	Time relative to start point	Time interval	Position
0	$65ns$	$0ns$	$0ns$	Input side of Input Port
1	$125ns$	$60ns$	$60ns$	Inside Input Port, after elastic FIFO
2a	$215ns$	$150ns$	$90ns$	After pipeline, channel request transmitted
2b	$245ns$	$180ns$	$30ns$	Positive flow control signal received, transmission starts
3	$255ns$	$190ns$	$10ns$	After CSU
4	$265ns$	$200ns$	$10ns$	After Output Port

The total time for a packet part to pass through a switch (with no collisions) is $200ns$ of which the time through the elastic FIFO and through the pipeline is 75%.

15.3 The schematic for four 8×8 channel switches.

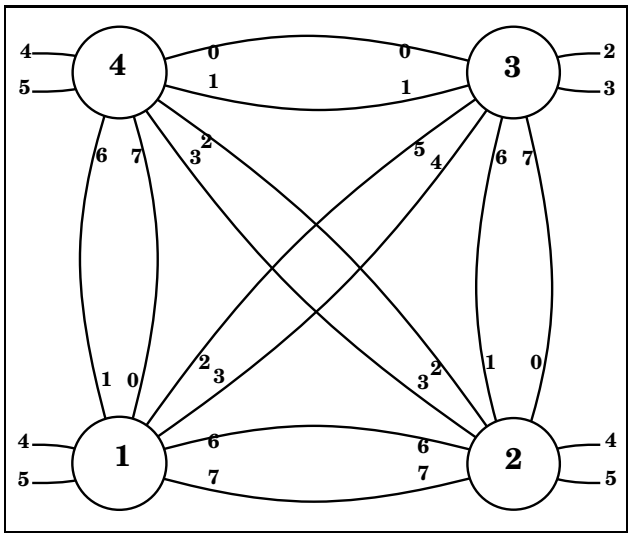


Figure 15.5: A network consisting of four 8×8 switches.

Figure 15.5 shows a small network consisting of four $8\times$ 8 channel switches. The figure shows the switch numbers and the channel numbers for all 4 switches. The switches have been connected with two channels between each pair of switches. Each switch has also two channels for external connections.

Figure 15.6 shows the SGE representation of the 8×8 channel switch. The position of the switches corresponds to figure 15.5.

15.4 Simulation results for four 8×8 switches.

15.4.1 Packet passing through five switches.

Figure 15.7 shows the simulated path for a packet transmitted through 5 switches. The packet enters on channel 4 of switch 1, continues through channel 7 to switch 2, through channel 0 to

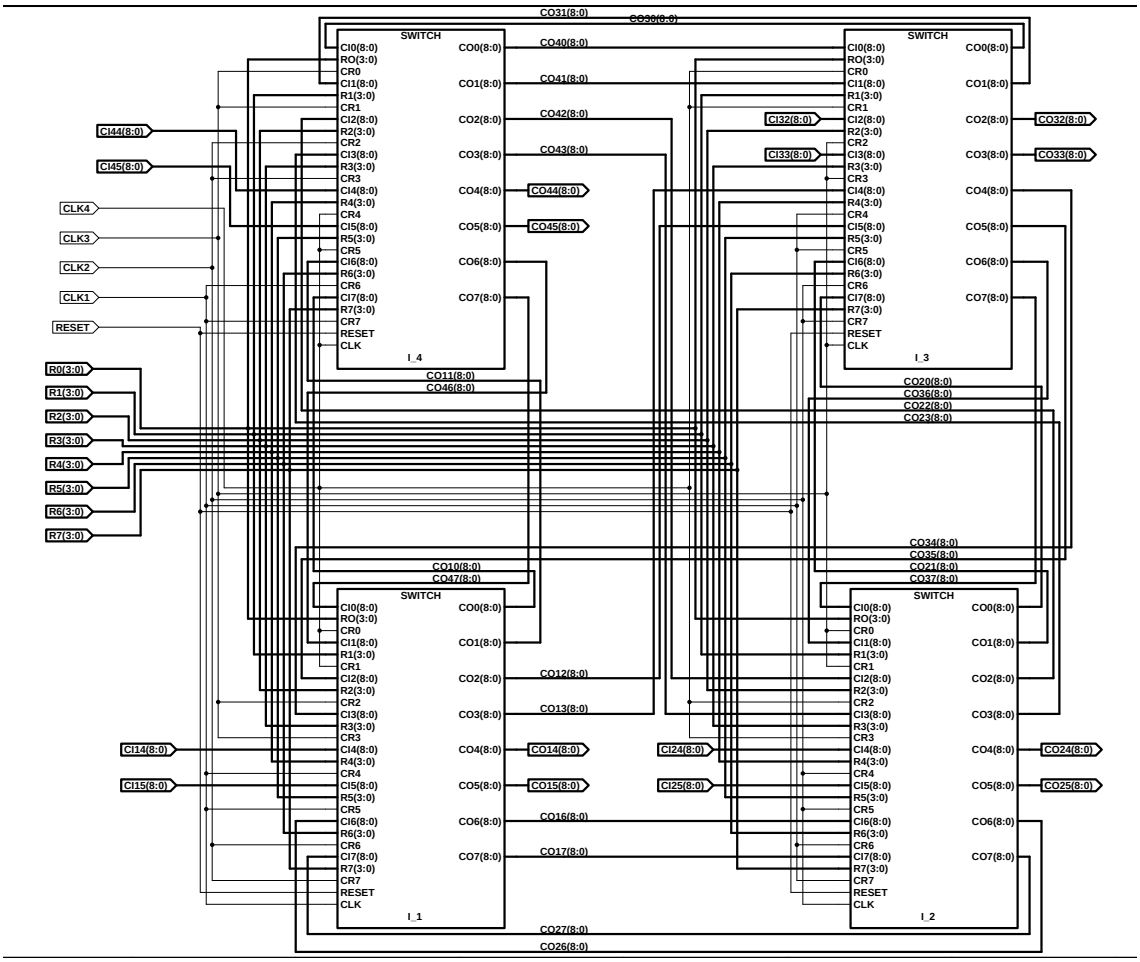


Figure 15.6: The SGE representation of a network consisting of four 8×8 *channel switches.*

switch 3, channel 1 to switch 4, channel 6 to switch 1 and finally leaves the network through channel 5 of switch 1.

Figure 15.8 shows a simulation where a packet passes through the 6 channels. The packet can be identified by the "noise" on each bus row. We see that most of the time the bus rows contain the value "010". This is the *idle* symbol with a positive flow control value for the opposite direction of each channel.

15.4.2 Two packets routed to the same output channel.

Figure 15.9 shows the routing path for two packets going to the same output channel. The intention of this simulation is to see the negative flow control signal for the blocked packet. The first packet enters channel 4 on switch 1, is forwarded on channel 6 to switch 2, through channel 1 to switch 3 and finally out on output channel 3 of switch 3. The second packet enters channel 5 of switch 1, continues through channel 7 to switch 2, and is forwarded from switch 2 to switch 3 through channel 0. Also the second packet is routed for output channel 3 of switch 3, but it enters the switch a little later than packet one. Thus, it has to be stored in switch 3 until the

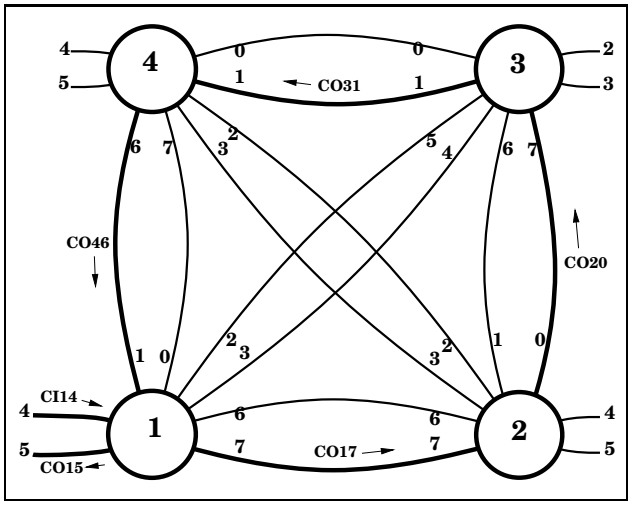


Figure 15.7: A packet passing through 5 switches counter-clockwise.

transmission of the first packet is finished.

Figure 15.10 shows the simulation curves for the two packets through the network. As for the other wave figures, the clock and the reset curve are at the top of the figure. In the remaining part of the figure each channel is divided into groups of two rows. The upper row marked by a **F:** in front is the channel in the direction forwarding the packet, while the lower row marked by a **R:** is the part of the channel carrying the flow control signal in the opposite direction. The three upper channels (SW1 CH4, SW1 CH6 and SW2 CH1) (6 rows) are the path segments used by the first packet. The next channel (SW3 CH3) is the common output channel for the two packets. Then the three exclusive channels of the second packet path (SW1 CH5, SW1 CH7 and SW2 CH0) follow.

We see a value change[1]on most reverse channels (prefix **R**) in figure 15.10 when the packet transmitted in the forward direction (prefix **F**) is close to its end. This is due to that the packet length is a little larger than the size decided by the "almost full" mark. Thus, a negative flow control signal is transmitted backwards. Almost simultaneously the packet head is forwarded, giving space for more data and resulting in a positive flow control signal transmitted backwards. In the bottom row we can see that the second packet receives a negative flow control signal '000', since the next path segment is occupied by the first packet.

15.4.3 Conclusion

In this chapter correct flow control signalling between switches has been verified. The simulation also verifies correct logical signalling between the switch elements described elsewhere in the thesis.

[1]With the simulated patterns the reverse channels only have the combinations 000="start transmission" and 010="stop transmission". Every second change is to 000, while the others are to 010.

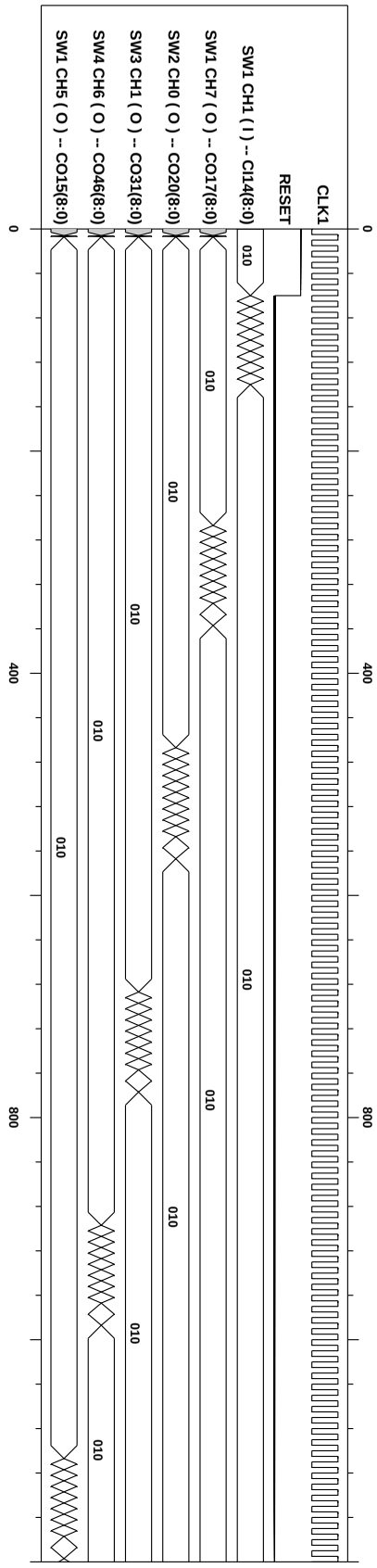


Figure 15.8: A packet passing through switch 1, 2, 3, 4 and 1.

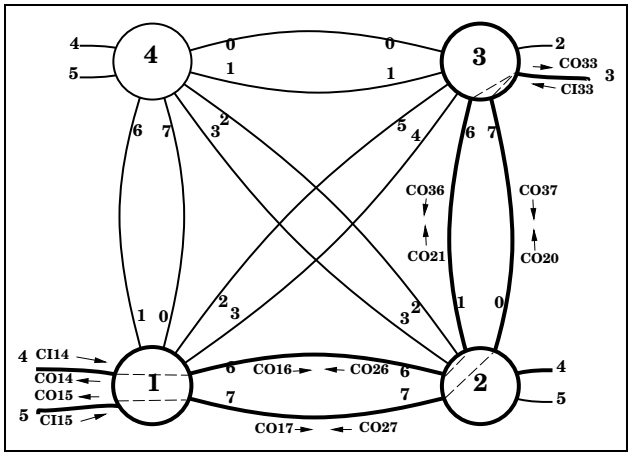


Figure 15.9: Two packets routed in parallel through switch 1 and 2 to the same output of switch 3.

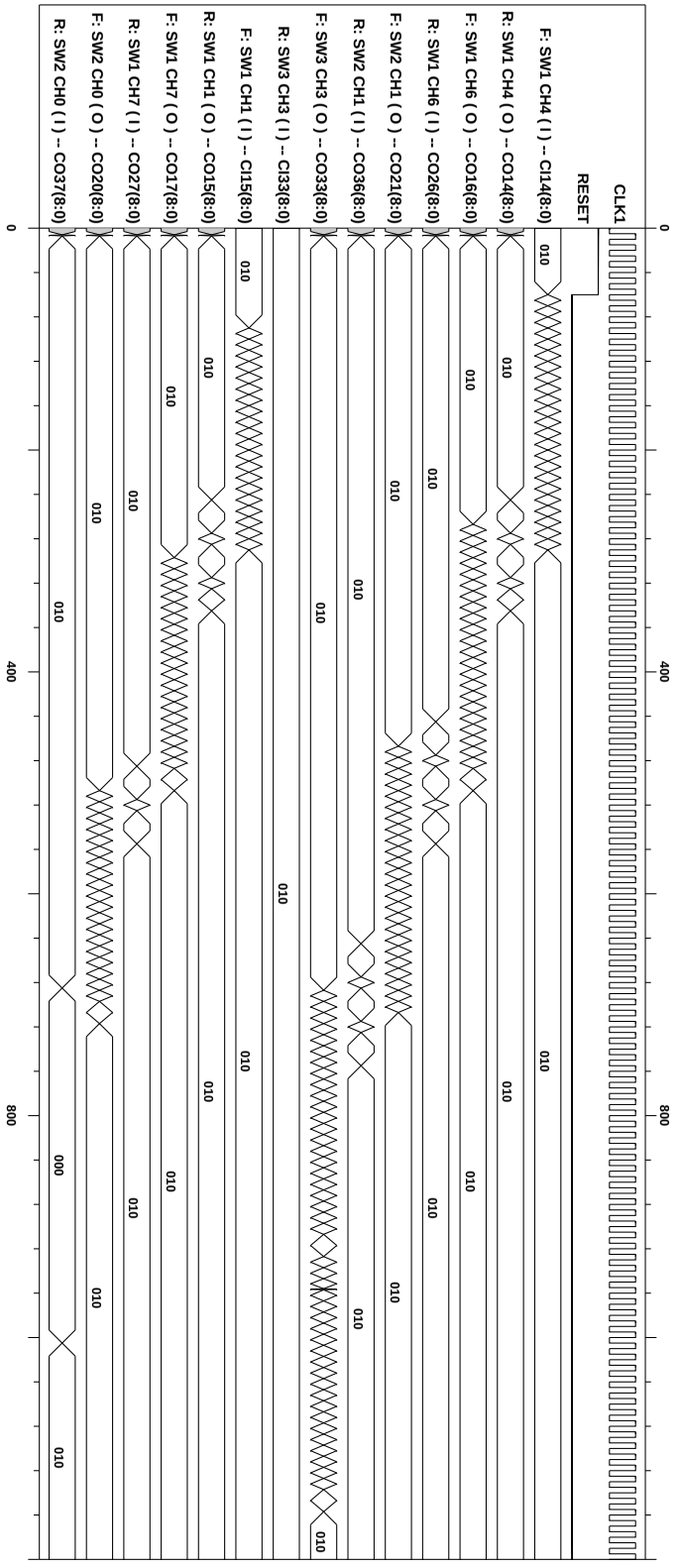


Figure 15.10: Two packets routed in parallel through switch 1 and 2 to the same output channel of switch 3.

PART 5

SWITCH IMPLEMENTATION

Chapter 16

Implementation of the CSU in CMOS and ECL

This chapter discusses the implementation of the CSU. Some of the SWIPP designs are described and discussed.

Several layout proposals for SWIPP switches have been made through the history of SWIPP. P.M. Baltzersen, a master student, proposed a store-and-forward switch [7] architecture. F. R. Karlsen, also master student, proposed a worm-hole architecture [54]. These switch architectures are no longer considered attractive, mainly because of low performance but also due to complicated physical construction. They will not be discussed further.

Experience with the new switch design has been gathered through simulations and designs by the author and undergraduate and graduate students under the author's supervision. In the following, two 4×4 channel switches which have been implemented and tested, a partially designed 16×16 channel switch (layout generated with auto-place-and-route tools), and a proposal for a full custom (manual place-and-route) switch will be presented.

16.0.4 The size of the CSU modules in number of transistors.

Before the different switch designs are presented, we will give an impression of the size of the CSU modules based on their transistor numbers. The exact number of transistors will depend on how latches and logical blocks are implemented. The numbers given in table 16.1 are for the 16×16 channel switch.

The CSU contains 3 types of modules: Crossbar, Arbiters and entrance module. The entrance module may be designed either for monocast or for both monocast and multicast. Table 16.1 also gives the number of transistors for four arbiters: one minimum size unfair arbiter, one speed-optimised unfair arbiter, one minimum size fair arbiter and one speed-optimised fair arbiter. The table gives the number of transistors and the number of latches. Two latch alternatives are suggested: a dynamic, high-speed latch with 12 transistors (figure J.2) and a low-speed latch with 24 transistors (figure J.1).

We see that a multicast module requires twice as many transistors as a monocast module. The cost of going from the simple unfair arbiter to the fast unfair one is similar to going to the slow fair arbiter. A minimum architecture with the smallest latch, monocast and a simple unfair

		L=12 tran.	L=24 tran.
Crossbar	8864	8864	8864
Arbiter with fixed arbitration sequence (13.6)	256L + 4608	7680	10752
As above but optimised on clock speed (D.6)	256L + 9472	12544	15616
Fair arbiter: LSLP (13.10)	512L + 7168	13312	19456
Faster fair arbiter: RDC ([63] Fig.7.7, pg. 76)	1024L + 7328	19616	31904
Monocast Entrance module	144L + 4480	6208	7936
Multicast Entrance module	464L + 7168	12736	18304

Table 16.1: Estimates of the number of transistors and the number of latches for the crossbar module, different arbiter (exit-module) elements and different entrance modules. The numbers are also shown for a latch with 12 transistors (L=12 tran.) and for a latch with 24 transistors (L=24 tran.).

arbiter requires 22 752 transistors. A maximum variant with the largest latch, multicast, and the fast fair arbiter requires 59 072 transistors, 2.6 times as many. A doubling of the bus width would not influence the arbiters or entrance blocks. Only the crossbar needs to be doubled.

Table 16.1 above gives an impression of the relative size of the logic implementation, although there is not a linear relationship between their size and the number of transistors they contain. The main differences are that the crossbar requires more space due to routing, while the arbiter can be made more dense. The arbiter contributes significantly to the size of the CSU and the number of transistors.

16.0.5 Clock method.

The switches described in this chapter are implemented with synchronous logic i.e. they contain latches clocked by a basic clock or by clocks derived from the basic clock. Plain synchronous logic demands that the period between the time at which a clock edge reaches different places within a circuit has to be small relatively to the clock period. Thus, at high clock speeds implementation of clock networks is a challenge and has to be carefully considered. An asynchronous implementation would reduce the clock problem by making it more local, but would add more circuitry. There is a third kind of switches: "fast" switches without latches. In SWIPP, the CSU contains arbitration and channel number decoding. Since the arbitration and decoders of SWIPP require latches, the latch-less switch architectures can not be used for SWIPP.

16.1 Two prototype 4×4 channel CSUs.

To get experience with implemented circuits, two smaller versions with 4 input channels and 4 output channels[1] were processed. This was done by two master students under the supervision of

[1] We decided to make a small circuit the first time to reduce cost and time, but also to make it simpler to verify the logical function. In a larger circuit, long signal and clock lines could add new problems. We wanted to reduce failures from this part in our first implementation. For the first prototypes we ended up with a 4×4 channel switch instead of the 16×16.

	Example ECL [41]	Example CMOS [92]	Our ECL [70]	Our CMOS [63]	New BiCMOS
Data rate	2Gbps	250Mbps	200Mbps	200Mbps	200Mbps
Bus width	1	1	5	5	5
Channel capacity	2Gbps	250Mbps	1Gbps	1Gbps	1Gbps
Number of channels	16	16	4	4	4
Time to establish connection	1)	1)	$15nS$	$65nS$	$65nS$
Connection mode	Monocast and broadcast	Monocast and multicast	Monocast and multicast	Monocast and multicast	Monocast and multicast
Connection establishing	One channel at a time	One channel at a time	All 4 simultaneously	All 4 simultaneously	All 4 simultaneously
Arbitration	External (not included)	External (not included)	Internal: Fixed unfair	Internal: LSLP	Internal: LSLP
Area pad ring included	$25mm^2$	$8.4mm^2$	$16.8mm^2$	$16.8mm^2$	?
Active area	1)	1)	$13.5mm^2$	$6.2mm^2$	?
Power	1984 mW	900mW	640mW	250mW	approx. 350mW
Technology	$1\mu m$ Bipolar	$1.25\mu m$ CMOS	$2\mu m$ BiCMOS	$1.5\mu m$ CMOS	$2\mu m$ BiCMOS
Price			890 ECU/mm^2	110 ECU/mm^2	890 ECU/mm^2

1) Values not available from paper.

Table 16.2: Comparison between two SWIPP CSU prototypes, two switch elements from the literature and a new CSU proposal for SWIPP.

the author. One circuit was designed in CMOS with a single-phase clocking strategy [112] by Jon Erik Kvarme [63]. Based on the author's ([77] [30]) growing interest in BiCMOS, another master student, Morten Moe ([70]), was supervised in ECL design in a BiCMOS technology. The logical architecture of the CMOS circuit and the ECL circuit are similar except for the arbitration block, so the main differences come from the differences in the technologies.

Table 16.2 above shows a comparison between two example crossbars reported in literature, one ECL and one CMOS and our implemented ECL and CMOS. The rightmost column shows a possible new design in BiCMOS where we take the best part from the ECL and the CMOS circuits. Both example switches are made for experimental purposes (not commercially available products).

Data rate:
The data rate is much higher for the example ECL compared to our ECL. One main reason for this is that we did not push the ECL speed very much in our design. Therefore it still has an unused potential. Another reason is that in mixed technologies like BiCMOS you have a reduced version of both technologies compared to the "pure" alternatives. The transistors in a pure bipolar technology will always be better than the bipolar transistors of a BiCMOS process. Similarly the MOSFET transistors in a BiCMOS process are better than the MOS transistors in a BiCMOS process.

We see from the table that the example CMOS and our CMOS circuit have similar speeds.

Bus width:
The example circuits have a bus width of one line, while our buses have a width of 5 lines.

Increase of bandwidth through larger bus width will give small increase in the control logic. The drawback is the increased number of pins used for this purpose which could have been used for a larger number of input and output channels (This would have demanded larger control logic).

Channel capacity:
The product of the bus width and the data rate per line gives the channel capacity.

Time to establish connection:
The time from a request-for-connection signal has entered the circuit until the connection has been established (given the output channel was unoccupied) is given in the 4th row of table 16.2.

Connection mode:
The meaning of monocast is that an input channel is connected to only one output channel.[2] Broadcast is ordinary broadcast where one input is connected to all 16 output channels. Multicast means that one input channel can send to any combination of output channels. The example circuits can only establish one new connection at a time. The reason for this is that the set-up commands use the same external bus for all channels. In our switch, the set-up commands are loaded through the ordinary input buses for each input channel. Thus, all channels can establish new connections simultaneously.

Arbitration:
For the example switches the arbitration is done outside the circuit. In both of our own circuits it is implemented on-chip. The fixed arbitration algorithm was chosen for the ECL circuit due to its smaller complexity. We wanted to allow more space for arbitration in the CMOS circuit. Thus the Last-Served-Lowest-Priority algorithm, which offers a fair treatment of the input channels, has been chosen.

Area with and without pad ring:
For our two circuits the number of pads controls the chip size. Not surprisingly the example CMOS circuit is smaller than our CMOS. The example circuit has a finer technology, smaller bus widths and no internal arbitration logic.

Power:
Both of our circuits use less power than the example circuits. One reason for this is the lower clock speed.

16.1.1 Testing of prototypes.

Both prototypes were produced and tested, the CMOS [62] more than the ECL. The measurements on both circuits were very unstable and it was clear that we had too little experience with measuring on high clock rate circuits, and that our test-equipment was not sufficient for these clock rates. After some trouble-shooting and modifications of the test setup, the circuit still gave different results for the same test pattern. We believe the cause of this is clock feeding. The clock strategy [112] requires sharp clock edges. With unstable clock feeding, we can expect behaviour as measured. If the average measured pattern is regarded as the true behaviour of the circuits, the results were encouraging. Except for temporary problems with decoupling connections, the behaviour was correct. Packets were routed correctly, and the arbitration circuits performed well. Resimulation of the CMOS circuit showed a design error resulting in unstable decoupling.

We believe that there are different ways to reduce the clock problem. The number of clock pads

[2] Several pairs may be connected simultaneously.

can be increased. Another solution is to distribute the clock signal from a clock buffer in the chip center. A third alternative is to have independent clock signals for each channel and route the clock signals close to their buses.

The circuits were only tested on 10 MHz. Due to problems with feeding sharp clock edges at this clock speed, increasing the clock rate to 100 MHz was not found valuable.

16.2 Switch layout generated with automatic place and route tools.

Appendix D gives some of the schemes for a 16×16 channel switch. The described switch does not support multicast. There are two common clocks: one for the arbitration logic and one for the remaining circuitry.

A layout of the switch circuit is shown in figure 16.1. The layout has been generated with a automatic place-and-route tool. An advantage of automatic place-and-route tools is the reduced time to generate the layout. One disadvantage is a somewhat larger area than what may be achieved through manual layout. Another disadvantage is the loss of control of routing. This may give unpredictable noise from neighbour wires. Lost control of clock routing may also give large clock skew. Generally clock lines have to be routed manually.

Anyway, the automatically generated layout normally gives a good indication of the circuit area, also for a manually designed circuit. Through area optimisation a smaller area may always be possible. The generated CSU layout has a width of $6623.1\mu m$ and a height of $6016.9\mu m$ giving a total area of about $40mm^2$.

16.3 Floorplan and local wiring for a manual layout.

Figure 16.2 shows a possible floorplan for a manually drawn layout for a 16×16 switch. The crossbar matrix is drawn as a column in the center of the figure. Each row contains one pair of entrance and exit blocks on the left side and one pair on the right side of the crossbar matrix. The input bus of each channel is routed through the entrance block to the crossbar matrix through which all input buses are routed vertically. The exit blocks containing the arbitration logic are placed closest to the crossbar matrix on its left and right side. Control lines for channel connections from an exit block are routed out to a private segment of the crossbar matrix. This segment contains the switching points between all input channels and the output channel of the exit block. These crossbar segments have triangular shape. As shown in figure 16.2 for each row two routing triangles are put together to make one square.

Outside the exit blocks we find the entrance blocks. Between the entrance and exit blocks is the routing channel. This routing channel contains the request and response lines between the entrance and exit blocks, since a dedicated request line is routed between each pair of entrance and exit blocks. Similarly response lines are routed from each exit block to each entrance block. These lines will occupy a significant part of the area. Thus, effort has been put into making this routing as dense as possible. A total of 256 request lines and 256 response lines are routed. This number of lines is needed when broadcast is supported. In monocast the same lines are required, but by moving some decoding logic from the entrance blocks to the exit blocks the number of wires between the blocks is reduced. With some of the monocast decoding logic in the exit blocks

each entrance block may have 5 or 8 "request" lines.

The connection lines between entrance blocks and exit blocks in figure 16.2 are placed in a circular routing channel[3] outside the crossbar and exit blocks. In the figure the routing channel between the entrance and exit blocks is on the left and right sides of the crossbar matrix. The routing channel is divided into a number of circular lanes. The lanes are numbered by increasing numbers with 0 closest to the exit blocks. A lane contains wires for different entrance and exit blocks in the different sectors. The different connections for the different sectors of a lane are denoted $a : b - b : a$, $c : d - d : c$, $e : f - f : e$ etc. In a multicast implementation, each lane contains two request and two response lines. The request and response lines in both directions for each pair of channel numbers are in the same lane. In the routing table below connections are given on the format $x : y - y : x$. The meaning of this syntax is that the section of the lane contains the following lines:
— the request line from entrance block x to exit block y.
— the response line from exit block y back again to entrance block x.
— the request line from entrance block y to exit block x.
— the response line from exit block x back again to entrance block y.

A proposal for how routing can be distributed in the different lanes is given in table 16.3.

[3] I.e. physical area where wires are routed.

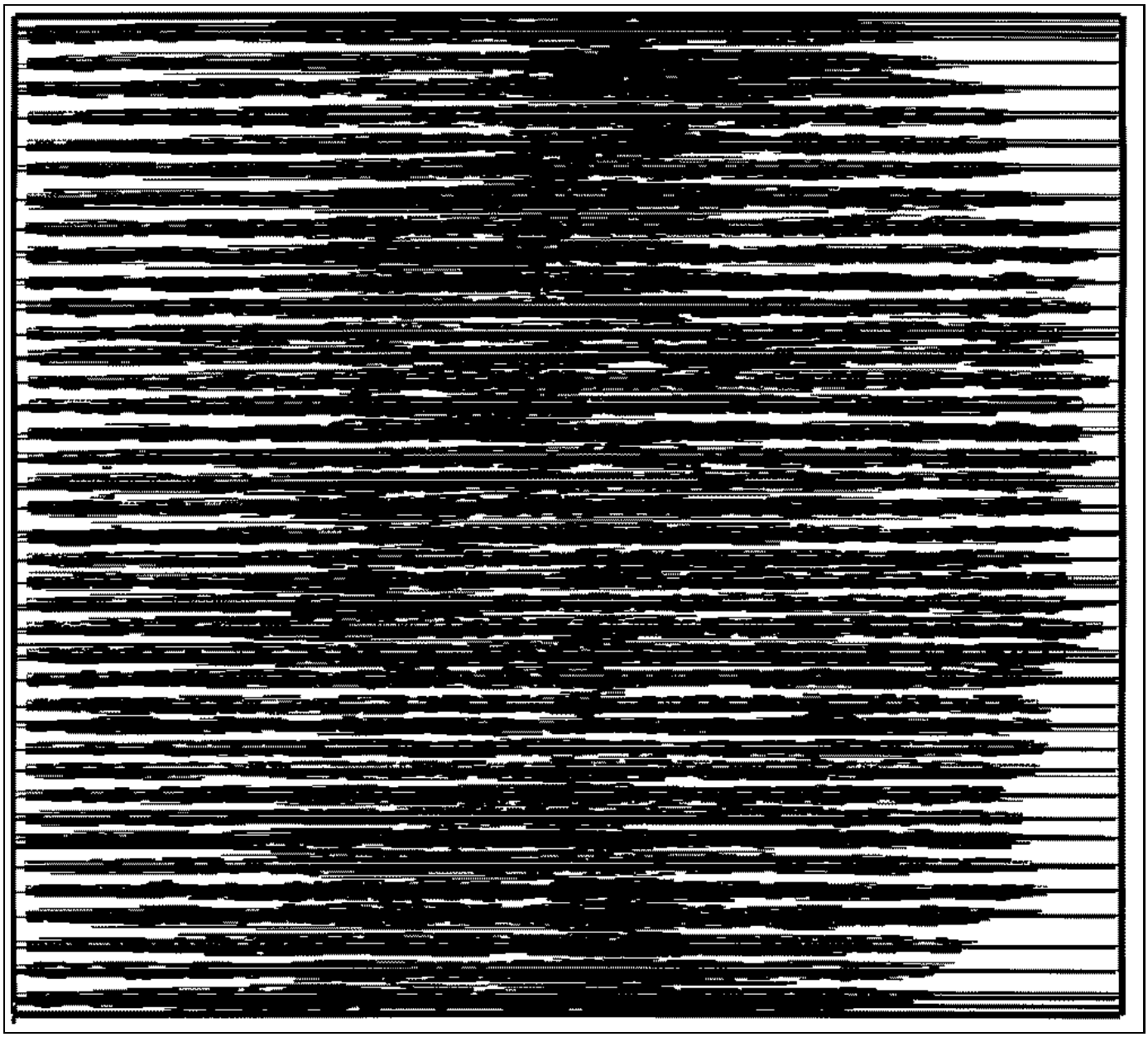


Figure 16.1: Layout of 16×16 *channel CSU generated by an auto-place-and-route tool. The CSU consists of a crossbar, fast unfair arbiters and a monocast entrance module. The horizontal dark lines on the figure are the 39 rows with library cells. The library cells are standard gates designed by the author in a* $1.2\mu m$ *technology. For higher clock rates, line propagation time and capacitive coupling between neighbour lines become significant. This requires change of routing rules and advanced auto-place-and-route tools for the highest frequencies or manual routing. The layout size of* $40mm^2$*, should be a good indication also for full custom solution.*

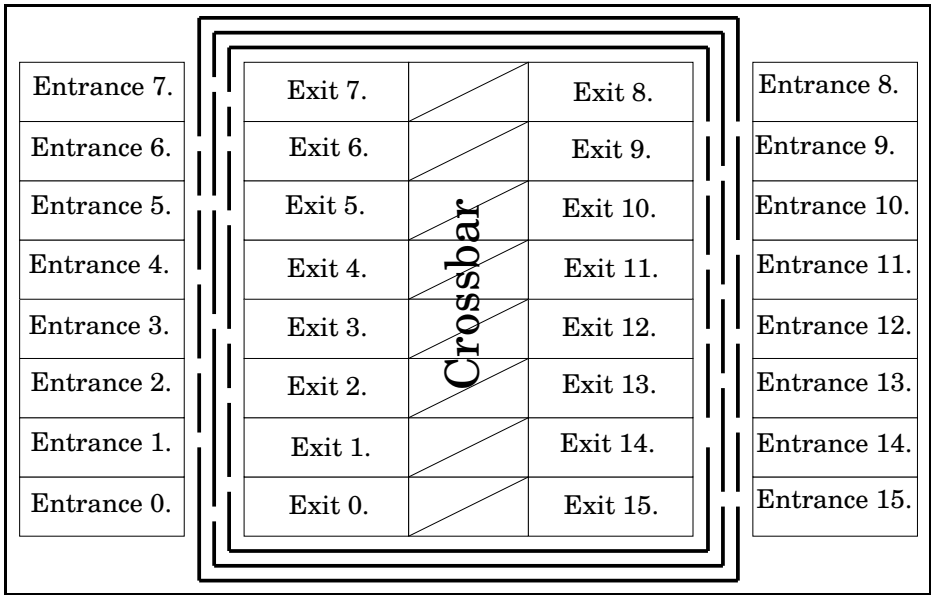


Figure 16.2: Floorplan for 16×16 *channel switch.*

The routing channel in fig. 16.2 is divided into several parallel lanes. The number of the lane is given in the leftmost column of table 16.3. The different sectors of a lane may contain routing between different channel pairs. The second column shows the number of channel pairs in a lane while the third and following columns give the identity of the channel pairs. The position of the connection points to other channels follow the same order internally in each entrance and exit block: increasing upward on the left side and downward on the right side.

16.3.1 Generation of the table.

The table in 16.3 was generated by a C program. The optimisation criterion was to reduce the number of lanes. The initial placement was optimised by exchanging connection pairs and groups of connection pairs and by placing connection pairs in different order. Four initial placement strategies were used: place longest connections first, place shortest connections first, place medium length connections first and random placement. Optimisation was running for several days with all four initial conditions. The best solution was one based on placing longest connections first. It was found in few hours. Continued optimisation for more than a week gave no further improvement.

16.4 A similar switch from the literature.

In [19] a 16×16 channel switch circuit is presented. For switching, the circuit uses a tree decoder very similar to the decoder used in our switch design. An important difference is that the switch in [19] does not contain address decoding and arbitration circuitry. Thus the circuit has a switching function and receives connection information through a control bus. Another difference is that the switch in [19] has a channel width equal to one line. Data are passing through the circuit without being clocked.

The bit propagation time through the [19] circuit is estimated to $6ns \pm 25\%$. When each data line was routed closely together with a private clock line, experiments have shown a signal skew of $\pm 0.1ns$ relative to the clock. When all 16 lines shared a common clock, the signal skew was $\pm 0.4ns$. The clock and signal skew influence the possible data rate.

The maximum data rate was found experimentally by narrowing a pulse to see how it was changed while it was transmitted through the switch circuit. The results are redrawn in figure 16.3.

Figure 16.4 shows simulated and experimentally verified bit rates for different technology scaling with the architecture described in [19]. An important statement in [19] is that crosstalk is primarily caused by packaging and output drivers. Internal crosstalk is believed to be insignificant.

16.5 The ECL and CMOS potential

Figure 16.5 from [40] (pg. 18) shows the propagation delay for ECL and CMOS. The horizontal axis is the minimum line width. The curves are calculated from speed measurements and typical load. We see that the delay decreases for shorter gate lengths. The curves show that when the minimum gate length gets close to $0.3\mu m$, the bipolar technology will be approximately three times faster than the MOS technology.

Connection pattern between entrance and exit blocks.							
Lane No.	Number of connections						
0	2	8:14 - 14:8	15:7 - 7:15				
1	2	6:12 - 12:6	13:5 - 5:13				
2	2	4:10 - 10:4	11:3 - 3:11				
3	2	2:8 - 8:2	9:1 - 1:9				
4	2	6:14 - 14:6	15:5 - 5:15				
5	2	4:12 - 12:4	13:3 - 3:13				
6	2	2:10 - 10:2	11:1 - 1:11				
7	2	0:8 - 8:0	8:15 -15:8				
8	2	7:14 - 14:7	15:6 - 6:15				
9	2	6:13 - 13:6	14:5 - 5:14				
10	2	5:12 - 12:5	13:4 - 4:13				
11	2	4:11 - 11:4	12:3 - 3:12				
12	2	3:10 - 10:3	11:2 - 2:11				
13	2	2:9 - 9:2	10:1 - 1:10				
14	2	1:8 - 8:1	9:0 - 0:9				
15	3	0:7 - 7:0	7:9 - 9:7	9:15 - 15:9			
16	3	5:7 - 7:5	7:13 - 13:7	14:4 - 4:14			
17	3	3:5 - 5:3	5:11 - 11:5	12:2 - 2:11			
18	3	1:3 - 3:1	3:9 - 9:3	10:0 - 0:10			
19	3	1:7 - 7:1	7:10 - 10:7	11:0 - 0:11			
20	3	0:6 - 6:0	6:10 - 10:6	10:15 - 15:10			
21	3	5:9 - 9:5	9:14 - 14:9	15:4 - 4:15			
22	3	4:8 - 8:4	8:13 - 13:8	14:3 - 3:14			
23	3	3:7 - 7:3	7:12 - 12:7	13:2 - 2:13			
24	3	2:6 - 6:2	6:11 - 11:6	12:1 - 1:12			
25	4	0:5 - 5:0	5:10 - 10:5	10:11 - 11:10	11:15 - 15:11		
26	4	4:9 - 9:4	9:10 - 10:9	10:14 - 14:10	15:3 - 3:15		
27	4	3:8 - 8:3	8:9 - 9:8	9:13 - 13:9	14:2 - 2:14		
28	4	2:7 - 7:2	7:8 - 8:7	8:12 - 12:8	13:1 - 1:13		
29	4	1:6 - 6:1	6:7 - 7:6	7:11 - 11:7	12:0 - 0:12		
30	5	1:5 - 5:1	5:6 - 6:5	6:9 - 9:6	9:12 - 12:9	13:0 - 0:13	
31	6	0:4 - 4:0	4:5 - 5:4	5:8 - 8:5	8:11 - 11:8	11:12 - 12:11	12:15 - 15:12
32	5	3:4 - 4:3	4:7 - 7:4	9:11 - 11:9	11:14 - 14:11	15:2 - 2:15	
33	6	2:3 - 3:2	3:6 - 6:3	6:8 - 8:6	8:10 - 10:8	10:13 - 13:10	14:1 - 1:14
34	4	2:5 - 5:2	10:12 - 12:10	12:14 - 14:12	15:1 - 1:15		
35	4	1:4 - 4:1	4:6 - 6:4	11:13 - 13:11	14:0 - 0:14		
36	3	0:3 - 3:0	12:13 - 13:12	13:15 - 15:13			
37	4	0:2 - 2:0	2:4 - 4:2	13:14 - 14:13	14:15 - 15:14		
38	2	1:2 - 2:1	15:0 - 0:15				
39	1	0:1 - 1:0					

Table 16.3: A table of how entrance and exit blocks may be grouped into lanes for a minimum total area.

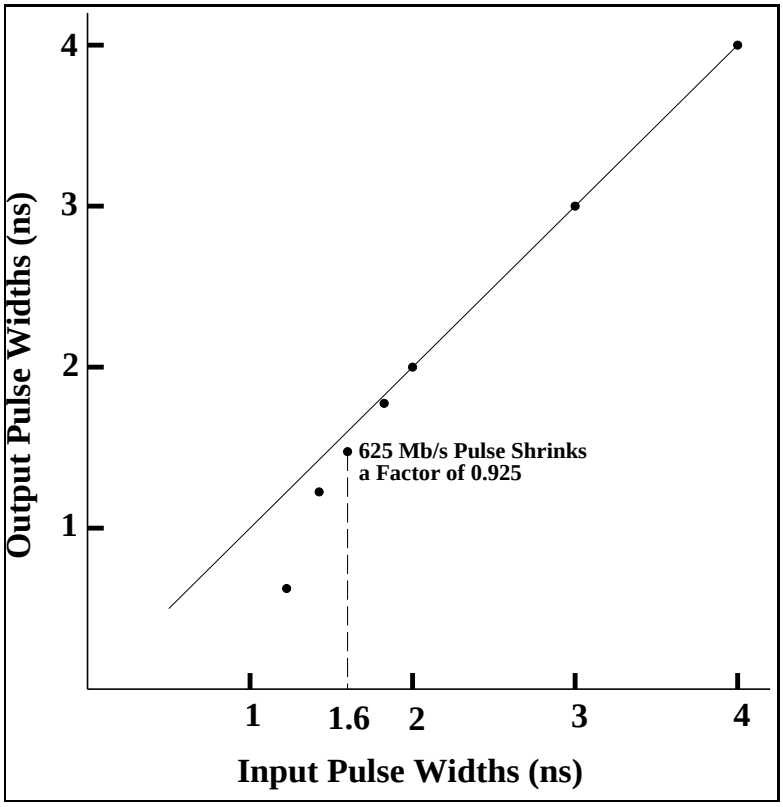


Figure 16.3: Experimental switching of narrow pulses in a 1.2μm technology. Pulse shrinking limits the bit rate. The figure shows that 1.6ns wide pulses are reduced at the output to 92.5% of the input size. A 1.6ns wide pulse corresponds to a bit rate of 625 Mb/s. Redrawn from [19].

Conclusion

The BiCMOS technology still has unused potential. The logic can be driven at higher clock speed, so our figures do not give representative characteristics of the technology. A mixed technology as BiCMOS has to compromise on both sides. Thus, bipolar transistors will be better in a pure bipolar process than in a mixed process. The situation for MOS transistors is similar.

The performance of CMOS circuits will increase, due to finer line widths and better architecture. The bipolar and BiCMOS processes are still much more expensive than the CMOS process. New CMOS processes can operate on high clock rates and allow a higher number of transistors per area unit (as discussed in connection with table 4.2). For bandwidth in communication, CMOS may compensate some of the missing clock rate compared to bipolar transistors through increased parallelism.

Figure 16.6 shows the layout of the 4×4 channel ECL prototype ([70]) discussed earlier in this chapter. The 4×4 channel CMOS prototype ([63]) discussed is shown in figure 16.7.

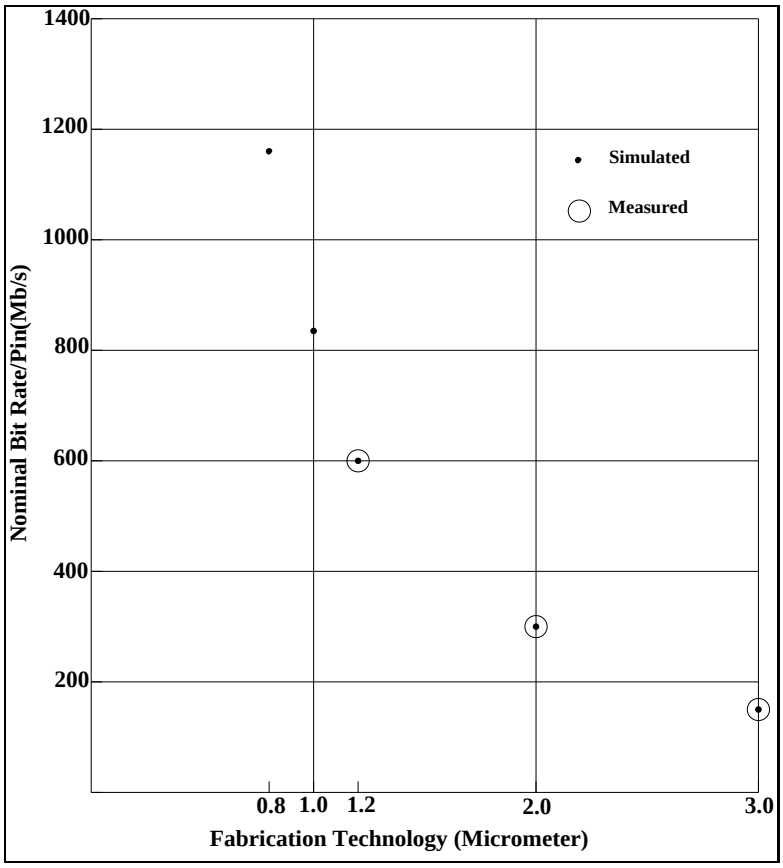


Figure 16.4: Simulated and experimentally found bit rates per line for technology scaling between $0.8\mu m$ and $3.0\mu m$. Redrawn from [19].

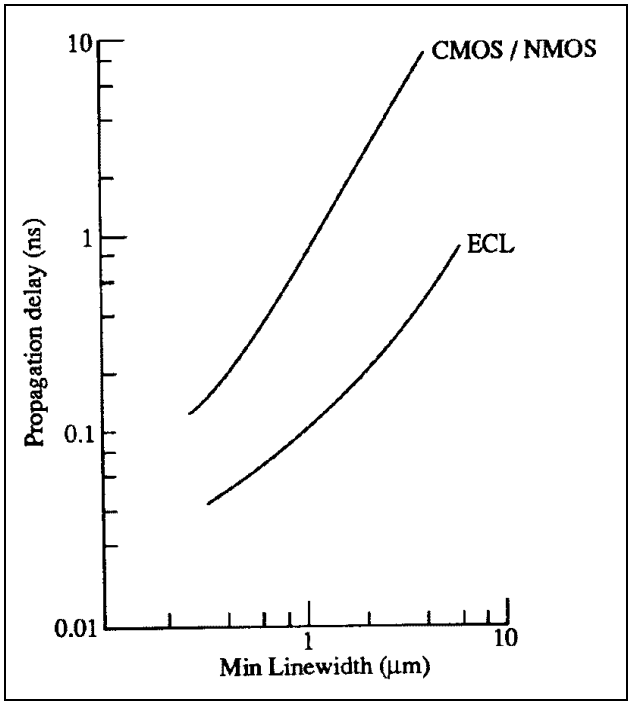


Figure 16.5: Speed for ECL and CMOS as a function of process quality.

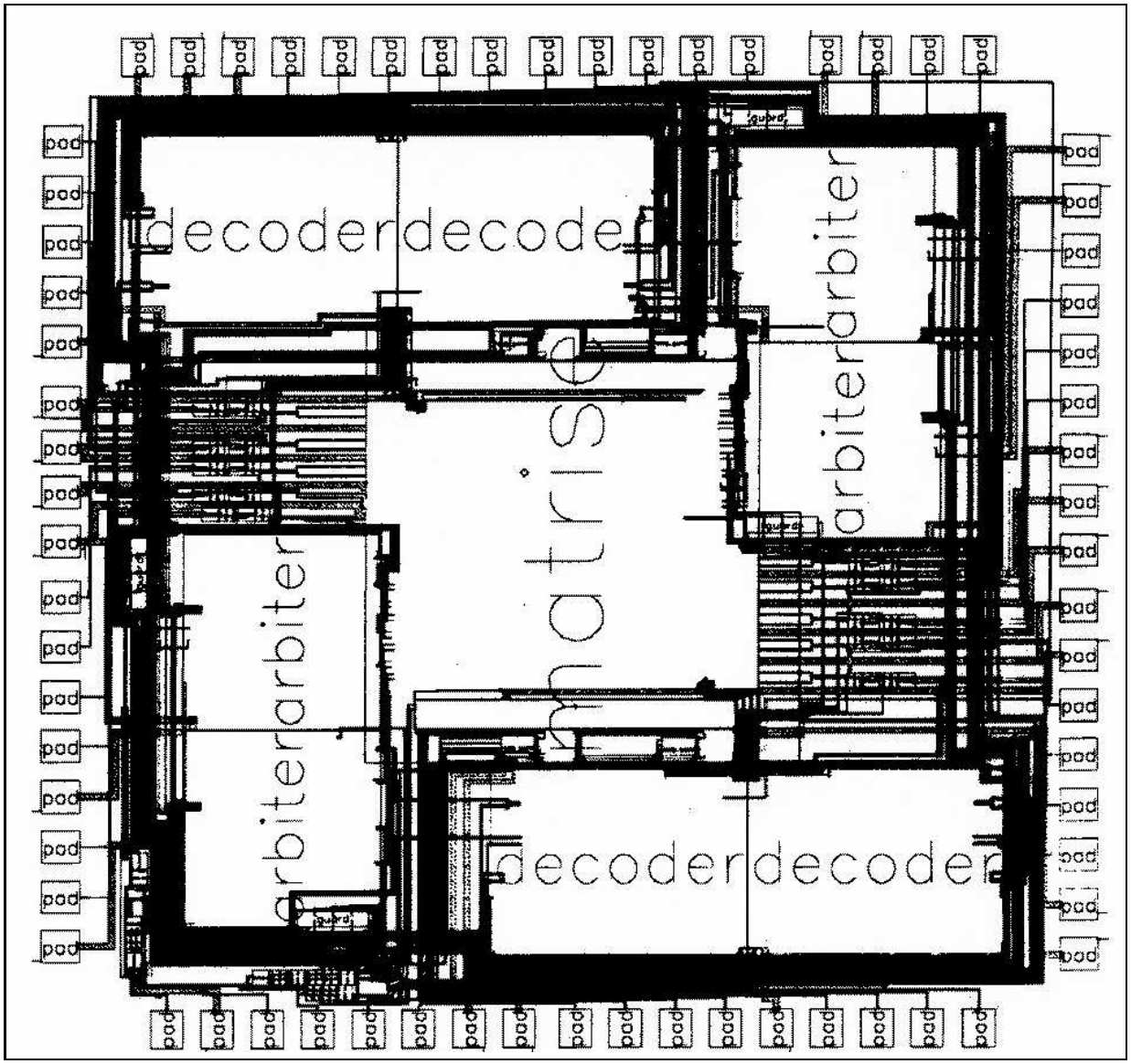


Figure 16.6: The figure shows the layout of the 4×4 channel ECL switch circuit.

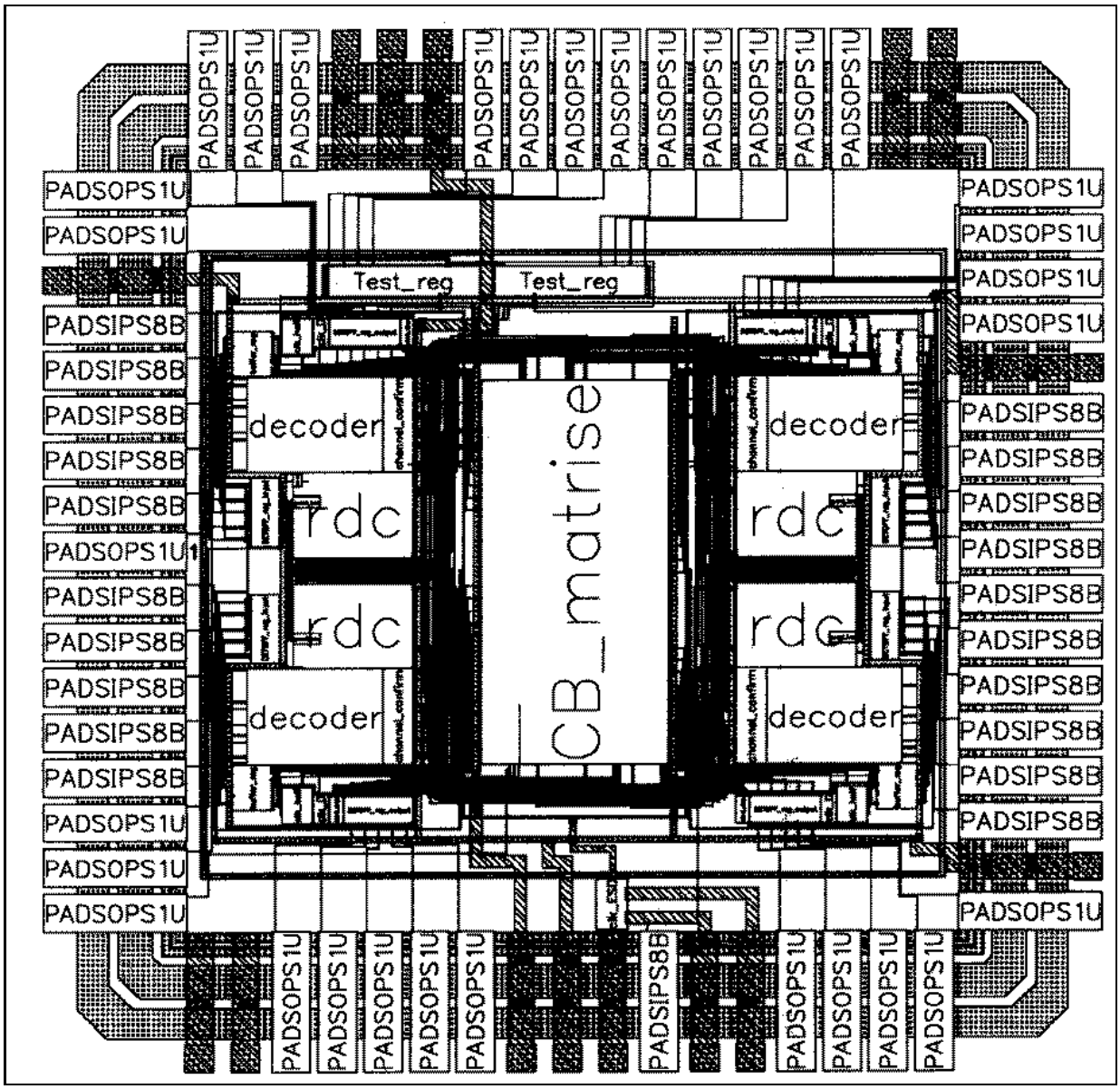


Figure 16.7: The figure shows the layout of the 4×4 channel CMOS switch circuit.

Chapter 17

Implementation of the Input and Output Port

This chapter describes the implementation of the Input and Output Port. The Input and Output Ports have been described at a functional level in chapter 10. Details of the implementation description are given in Appendix B. Some layouts are given in Appendix G.

17.1 Introduction

As stated before, the layouts have been done over several year. Thus, some lay-outs have not been designed with todays smaller geometrical rules. Also different effort has been put in making the layouts dense. With the lay-outs done and for the technologies they have been implemented in, it is not possible to intgrate the switch on one circuit. Therefore, the present description of the switch node is divided into different blocks. From the work of Pål Baltzersen and Frode Karlsen (see [7] and [54]) we made the conclusion that each circuit should have one central switch fabric and pre- and post- processing be pushed to neighbour chips. The preprocessing blocks are gathered in the Input Port while the post- processing is performed in the Output Port. As stated in chapter 10 the elastic FIFO is a part of the Input Port. This author has used the opportunity to let some FIFO architecture proposals be implemented as part of projects in the integrated design course: *IN241 VLSI konstruksjon.* A solution has also been implemented by one of the master students [86] supervised by this author. Implementation of the elastic FIFO is further described in chapter *18, Implementation of the elastic FIFO.*

17.2 The Input Port

17.2.1 Recapitulation of the main functions.

As discussed in chapter 10, the main functions of the Input Port are as follows:
- reading address nibble from packet,
- generating channel request for the Central Switch Unit,
- intermediate buffering of incoming packets,
- forwarding packet data depending on flow control signal, and

- updating and changing the symbol sequence in a packet.

The logic for these functions is located between the FIFO and the output bus of the Input Port.

The internal clock frequency is 100 MHz.

17.2.2 First versions of the Input Port

Our first version had a state machine with 6 states and 32 transitions (Appendix F). The state machine had access to the first two symbols of the FIFO. Parts of these two symbols were routed together with fixed values to the output bus of the Input Port.

It was found during the design phase that it was not possible to fulfill the 100 MHz requirement. The delay of the signals on the way from latch outputs through the combinatoric network to the latch inputs was for many paths between 10 and 20 ns. A number of redesigns did not make these delay values significantly smaller, so it was clear that the task had to be handled in another way.

17.2.3 Input Ports supporting broadcast and priority classes

We divide broadcast into two strategies: Broadcast is performed only by the last switch that a packet is passing through, or broadcast is done by several of the switches. Multiple level broadcast can generate circular packet paths or other phenomena which may degrade the performance of the network considerably. We did not find any good solution for how multiple level broadcast should be handled. Neither did we have any clear idea of what need for broadcast the possible applications would have. We decided to leave this topic for a future study.

Another subject for study was the possibility to give packets different priority. A number of applications would benefit from this. Some solutions for how this could be done have been worked out, but we found them too complicated for the first versions. The priority could be taken into account when packets from different input channels compete for the same output channel. This requires a more complicated arbiter design. If the arbiters grow too large, they have to be positioned outside the CSU, which would increase the time to settle an arbitration task. The priority could also be taken into account between packets arriving on the same input channel. To simplify buffer management, the packets should have fixed size.

17.2.4 Present version of the Input Port

In the last version, the packet interpreting and manipulation functions are distributed along a pipeline. This makes the logic larger, but now, as shown in appendix B, it satisfies the 100 MHz requirement within good margins. Figure 17.1 shows the pipeline.

The elastic FIFO is on the left side outside our figure. The CSU is on the right side outside the figure and is loaded by the **I**-register. The capital letters are the 9 registers of the pipeline. The number on top of each register is the number of bits.

As we can see from the figure the pipeline is divided into 4 sections: **the decoder section**, **the a4-section**, **the nibble manipulation section** and **the output section**.

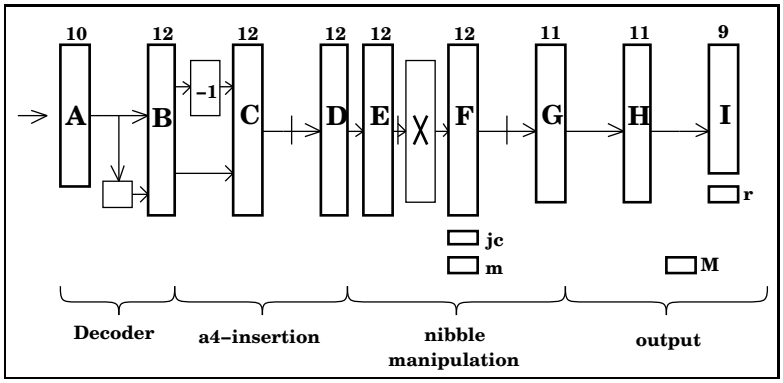


Figure 17.1: The control logic of the Input Port implemented as a pipeline.

The decoder section analyses the incoming symbol to see whether it is any of three specific control symbols.

The a4-insertion section updates the **jump counter** and prepares a request signal for the CSU.

The nibble manipulation section removes, inserts and changes the order of nibbles in the packet.

The output section controls shifting of data. It communicates with the switch fabric and shifts data out of the FIFO and through the pipeline. It generates *end-of-packet* signals to the switch fabric when the connection is to be broken.

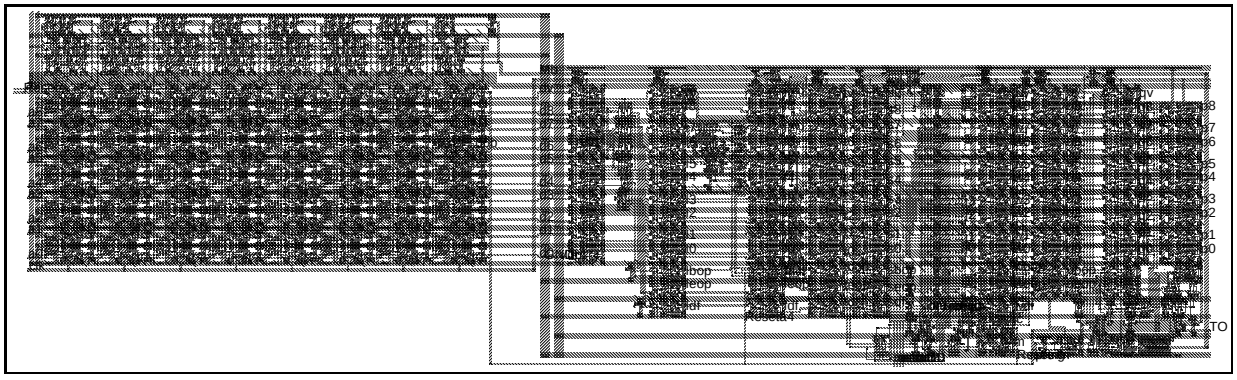


Figure 17.2: Layout of the Input Port.

Figure 17.2 shows a layout of the Input Port. The pipeline is on the right half with the same orientation as in the previous figure. The Input Port is shown with an 8×9 bit synchronous FIFO on the left side. This FIFO is only used for testing of the pipeline. In real use an elastic buffer such as an asynchronous FIFO or a one- or two-port RAM with transfer logic between clock regions should be used.

The Input Port as given in the figure consists of 3 796 transistors. In a $1.0\mu m$ process with $1\lambda = 0.5\mu m$ the size would be $1542.5\mu m \times 459.5\mu m = 0.71mm^2$. As we can see from the figure

the pipeline covers about half of the layout.

Conclusion
The pipeline has been designed and simulated (with parameters from an Orbit $1.6\mu m$ technology) both by using electrical and logical simulators. It operates at 100 MHz with good safety margin. All signals are stable within $2-3ns$ indicating that clock rates in the area of 200 MHz should be possible.

17.3 Output Port

The Output Port is described in chapter 10. The only function of the Output Ports is to extract and insert flow control signals.

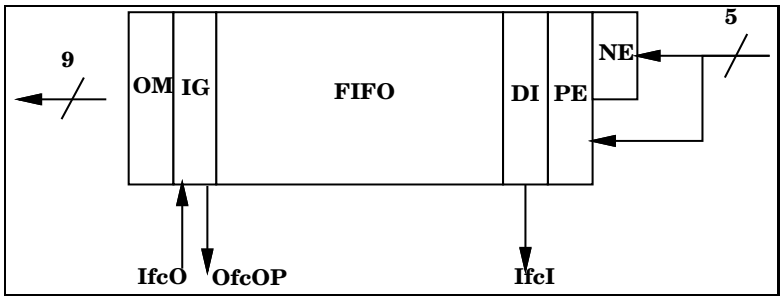


Figure 17.3: The blocks of the Output Port.

Figure 17.3 shows a sketch of the Output Port. The data stream from the CSU arrives on the right side. It passes through two registers which double the bus width and halves the clock rate by placing two and two nibbles together. The flow control signal from the crossbar is extracted from the incoming data stream and forwarded to the Input Port. At the other end of the Output Port, flow control signals from the Input Port are inserted into the stream.

As described in chapter 10, a FIFO is necessary for the insertion of control symbols with flow control information into the data stream. Insertion of signals results in that the subsequent data are delayed.

Figure 17.4 shows a layout for the Output Port. In a $1\mu m$ process the size will be $327\mu m \times 840\mu m = 0.27mm^2$. As we can see from the figure, the FIFO circuit covers the greater part of the area of the Output Port.

The layout of the Output Port is described in greater detail in appendix B.

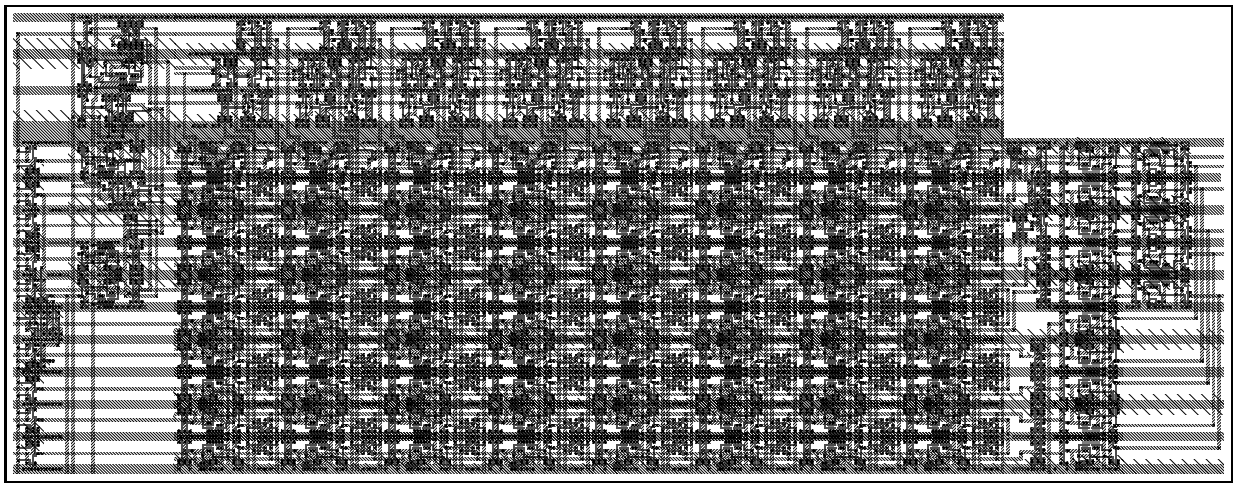


Figure 17.4: The layout of the Output Port.

Chapter 18

Implementation of the elastic FIFO

This chapter describes and discusses three ways of implementing an elastic FIFO.

18.0.1 The position and function of the FIFO

The elastic FIFOs are situated inside the Input Ports. The FIFOs are the main storage elements in the SWIPP network. They are used to store data packets when data can not be forwarded either due to the output channel being occupied or because there is no storage capacity available down stream. The term "elastic" in this case means that the input and output sides of the FIFOs have to operate on different clocks.

18.1 The asynchronous FIFO based on Sutherland's methods.

The logic of this FIFO is based on guaranteed delays ([105] [104]) which assure the sequence between signals. Thus, it can be guaranteed that a signal is generated after data have been safely stored. The logic uses one pair of request and acknowledge signals on the input side and one pair on the output side of the FIFO. On the input side, a transition on the request line indicates that there are new valid data on the input bus. The logic block generates a transition on the acknowledge line after the data have been read. On the output side, a transition on the request line indicates new data. After the data have been read the reader will respond with a transition on the acknowledge line.

One of the master students supervised by this author, Per Torstein Røine, has implemented an elastic FIFO in this design method [86]. A photograph of his finished chip is shown in figure 18.1. The design parametres are given in the table below.

Total size:	$26mm^2(4.2mm \times 6.2mm)$
Active area:	$9.5mm^2$
Technology:	$1.5\mu m$ CMOS
Input channel:	9 lines asynchronous
Output channels:	2×9 lines synchronous
Max frequency (corresponding):	100 MHz
Storage capacity:	75 words a 9 bit
Power consumption:	0.1 - 0.4 W at 50 MHz

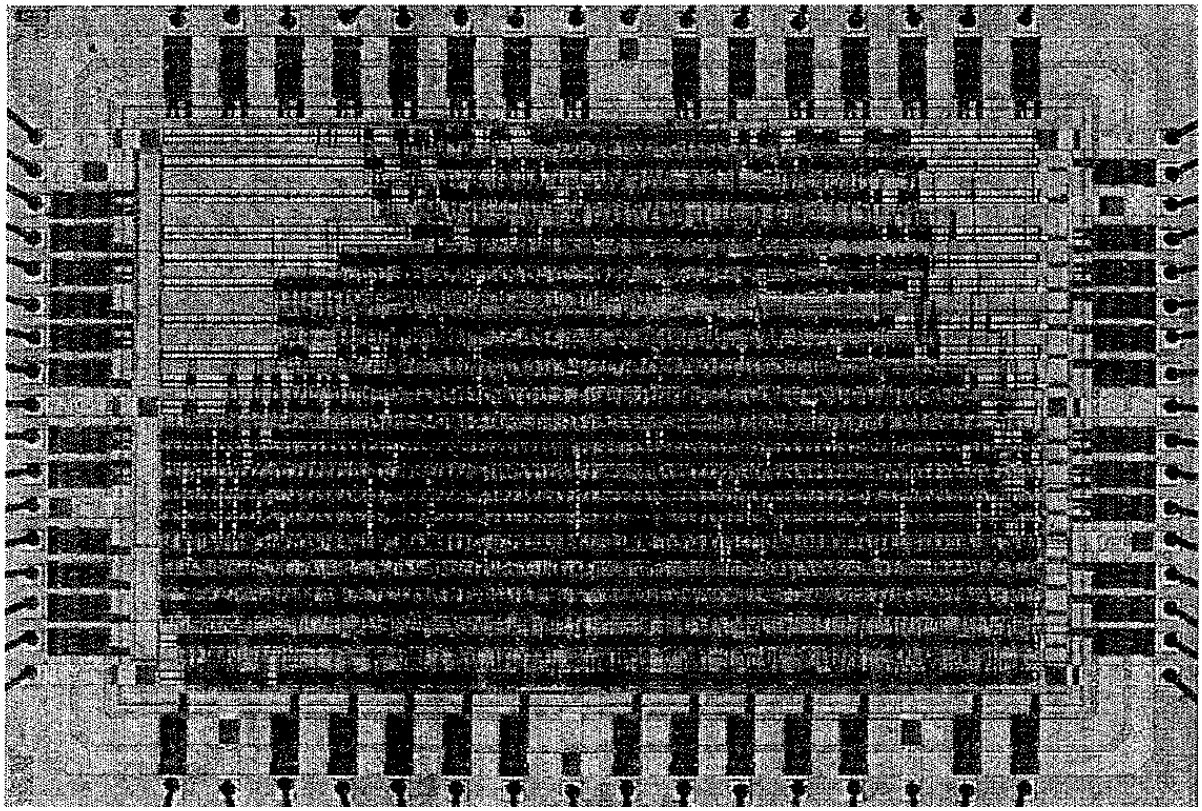


Figure 18.1: The figure shows the asynchronous FIFO designed by P.T. Røine for his master thesis.

18.2 Elastic FIFO built on a two-port RAM

This section describes this author's own idea of how an elastic FIFO can be built from a two-port RAM. Architectures built on ordinary two-port RAMs will in many cases be a good alternative due to their smaller area per. memory bit.

The architecture of the two-port RAM is given in figure 18.2. The incoming and outgoing data streams are operating on a clock speed of 100 MHz. The two independent ports of the two-port RAM operate each on 25 MHz clocks. Due to this difference in clock speed, the word length of the RAM is 4 times as large as the incoming and outgoing data stream. The figure shows how the high-speed words are placed in the different slots of the RAM word on the left side. On the right side the outgoing words are taken from different slots of the RAM word. This solution gives a somewhat higher propagation time through the FIFO than other FIFO implementation solutions. The advantage is the smaller area per bit compared to other alternatives. Bandwidth equal to that of other solutions is achieved through parallel treatment of data.

When only a small part of the RAM is filled, data words are written to the RAM if at least one slot has been filled. This takes place with the rate of the 25MHz write clock. There are two reasons for doing it this way. It will make the pass-through latency as small as possible when there are few data in the RAM block. It will also assure that all data are forwarded to the output side.

When the amount of data in the RAM buffer increases, words will be written to the RAM block

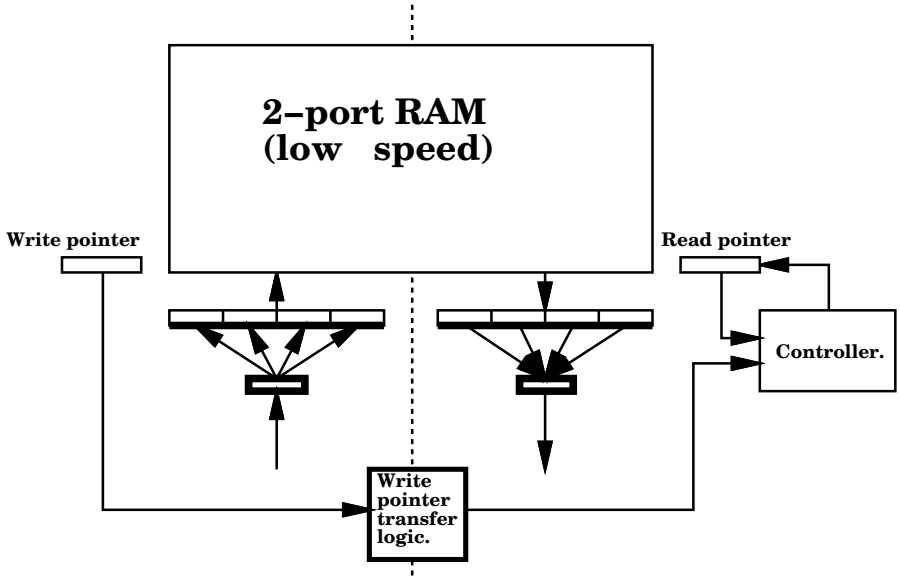


Figure 18.2: Elastic FIFO built on a two-port RAM. The left side of the dashed line is the write (input) part while the right side is the read (output) part. Hence the dashed line gives the border between the two clock regions. The RAM core is common. The high speed registers are drawn with thicker lines.

only when all slots have been filled. This is done to attain dense utilisation of the RAM block.

18.2.1 The write pointer transfer logic and the read pointer.

To make sure that the output side always reads RAM data which are valid and stabilised, the write pointer is transferred to the output side. The read pointer can advance and read out data up to the copied write pointer value. The transfer of the write pointer is done by the write pointer transfer logic.

Figure 18.3 shows the logic used to transfer the write pointer from the transmitting side to the receiving side of the main buffer. An SR-latch is a central element in this write pointer transfer logic. The SR-latch is set only by the transmitting side and reset only by the receiving side. The output value from the SR-latch is read both by the transmitting and receiving sides by separate latches clocked by slow clocks. Occasionally the two latches reading the SR-latch, (the one on the transmitting side and the one on the receiving side), will clock the output of the SR-latch during a transition.[1] When this happens, these latches read a value between high and low. The two latches reading the SR-latch contain local feedback loops. These feedback loops amplify any deviation from the metastable potential[2], forcing the potential high or low. If the read potential is above the metastable potential, the output will be forced high while if the read potential is below, the output will be forced low. How quickly the potential is forced high or low depends on the small-signal gain-bandwidth product of the latches and how close the "caught" value is

[1] This will probably not happen simultaneously on both the transmitting and receiving sides.

[2] A simple definition of the metastabile potential u_M is that u_M is the potential u fulfilling the equation $u = H(u)$. Here $H()$ is the amplification function of one of the latches reading the SR-latch. Since different latches have different $H()$, their metastable potentials are different.

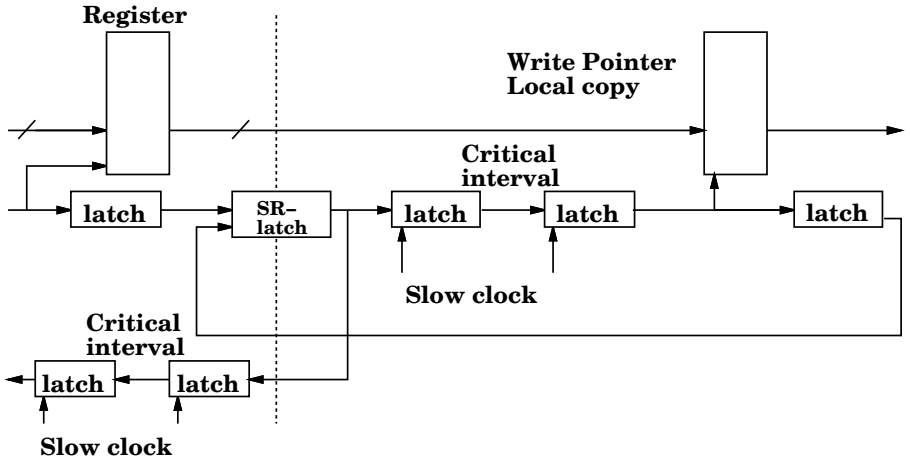


Figure 18.3: The figure shows logic for transfer of register data between clock regions. The dashed line shows the clock border. All clocks on the transmitting half (left side) are independent of the clocks on the receiving half (right side). The maximum cycle time is $2t_{ts} + t_t + 2t_{rs} + t_r$, where t_{ts} is the slow clock period on the transmitting side, t_t is the standard period on the transmitting side, t_{rs} is the slow period on the receiving side and t_r the standard period on the receiving side. t_{ts} and t_{rs} have to be long enough to reduce the probability of metastability to an acceptable level. The minimum time from a "write" into the transmitting register to a "read" of the same register is t_{rs}. The minimum time from a "read" to a "write" is t_{ts}.

to the metastabile point. In [86] chapter 7 an approximate expression for the MTBF[3] due to metastability has been developed.

$$\text{MTBF} \approx \frac{1}{f_S f_D \frac{u_E}{k}} e^{t_w/\tau} \tag{18.1}$$

Here f_S is the clock frequency of the reading latch and f_D is the frequency of transitions in the input signal. t_w is the time from the input signal is copied by the first latch until the first latch output is clocked into the following latch. This is the time available for the signal to stabilise at a defined potential. τ is the time constant of which the value approaches the high, alternatively low, potential[4]. τ depends on the implementation of the latch. In this model, an error is defined to have occurred if the potential after t_w, is still within a range $u_E/2$ above and $u_E/2$ below the metastable potential. The value k is the increase rate of the input signal at the transition through the metastable point[5] Example values are $f_S = 100MHz$, $f_D = 10MHz$, $u_E/k = 1ns$. From [86] we may use the value B to find $1/B = \tau = 262ps$. If the clock period is $t_w = 5ns$ (1/200 MHz) the MTBF is approximately $194s$. If the time to stabilise is increased to three clock periods ($t_w = 15ns$), the MTBF has increased to $2.3 \cdot 10^{11}$ years.

The metastability problem is illustrated in figure 18.4.

[3] Mean Time Between Faults

[4] $(1/\tau)$ is in [86] entitled an "escape factor" B

[5] The factor u_E/k is proportional to the raise/fall time of the input signal. If u_E is the potential between10 and 90 % of the full range signal, u_E/k is equal to the raise/fall time. If u_E is the potential between30 and 70 % of the full range we have that u_E/k is half of the raise/fall time.

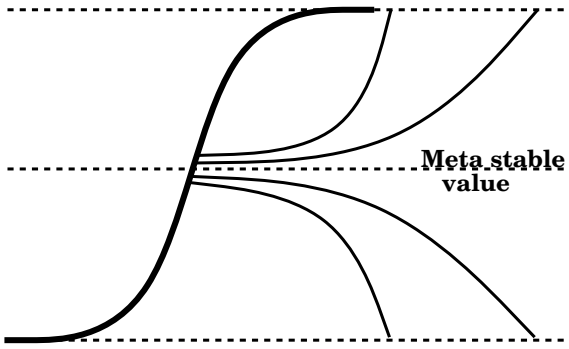


Figure 18.4: The metastability problem. A signal is clocked by a latch during transition. The potential of the caught value changes with time according to the equation $u(t) = u_M + \Delta u e^{t/\tau}$. Here u_M is the metastable potential and Δu the distance of the caught value from the metastable potential

When the output side of figure 18.3 has received a stable signal, the write pointer will be written into a local latch. Now the SR-latch will be reset and the input side may prepare a new write pointer for the output side. The SR-latch is set and reset alternatively. It will always be at least one clock period between a latch set and a latch reset.

18.2.2 The write pointer.

A side effect of the flow control system is that the write pointer never passes the read pointer i.e. that new values do not overwrite old unread values.

18.3 Elastic FIFO built on a one-port RAM.

Another possibility is to use a one-port RAM, where the whole RAM is running on the transmitting clock. The transfer logic between clock domains is on the output side of the RAM. In this solution the transfer register in 18.3 should contain the data. The amount of data in each transfer should be enough to occupy the following logic, until a next transfer can take place. The advantage of this method is the high bit density offered per area for the one-port RAM.

18.4 Comparison between the asynchronous FIFO and the two-port RAM.

Figure 18.5 shows the curves indicating the sizes of the two alternatives given above.

The values for the asynchronous FIFO are the exact values found by actual design. These sizes include some logic for extraction of control signals, which are not included in the curve of the two-port RAM. The curve of the two-port RAM contains only the core part[6]. Thus, the high speed

[6]The remaining part is expected to have an area less than $1mm^2$

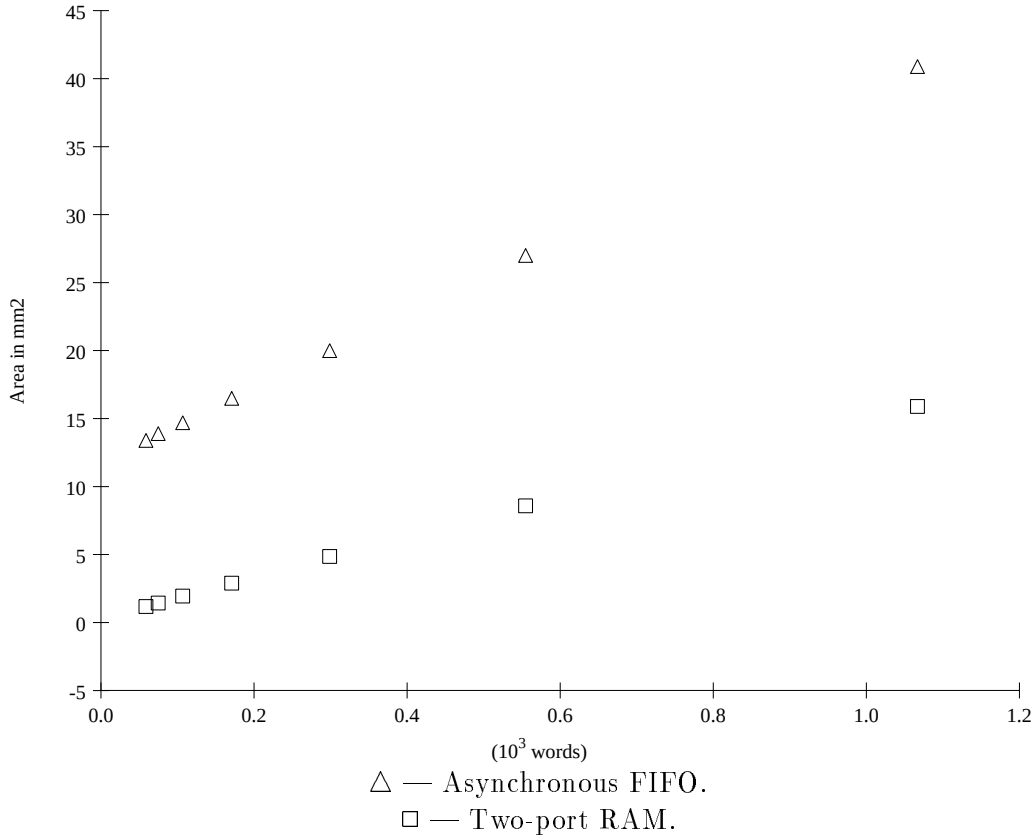


Figure 18.5: The upper values show the memory area for the asynchronous FIFO found by real design. The lower curve is for the core of a two-port RAM. This curve is estimated from a self-made 32-byte RAM. The values on the vertical axis are given in mm^2.

registers and the *write pointer transfer logic* are not included. This makes the difference look larger than it really is. A comparison of the core cell areas is given in table 18.1. The fall-through FIFO in table 18.1 is the FIFO shown with the Input Port in appendix B. The fall-through FIFO is also shown in figure G.1. An example of the 1-port RAM is given in figure G.7.

It is clear that a one-port RAM uses less area per bit than the asynchronous FIFO. The data amount to be stored is only depending on the rate with which new packets arrive, the packet lengths and the bandwidth capacity. Thus, if buffers at the input and output sides hide a slow core, the clock rate of the core will neither influence the bandwidth nor the amounts of data to be stored.

In his thesis ([86] pg. 76) the designer of the asynchronous FIFO compared the energy used by the asynchronous FIFO with the energy used by a two-port RAM. He found that his asynchronous FIFO uses 3.4 to 5.4 times as much energy as the two-port-RAM FIFO.

	Area for 9-bit word
1-port RAM	$8676\mu m^2$
Fall-through FIFO	$36621\mu m^2$
Asynchronous FIFO	$90747\mu m^2$

Table 18.1: Core cell areas for some FIFO architectures drawn in $1.2\mu m$ *technologies.*

Chapter 19

Implementation of the Optical Module

This chapter discusses different Optical Modules. A layout of the architecture presented in chapter 11 is given. The last part presents some commercially available alternatives.

A number of versions of different parts of the Optical Module architecture from chapter 11 have been designed. Under the supervision of this author, some implementation proposals have been designed by undergraduate students and one by the master student Manickam Rajeswaran Sivasothy [94]. The plots and numbers in this chapter come mainly from Sivasothy's work.

19.0.1 The encoder implementation by Sivasothy.

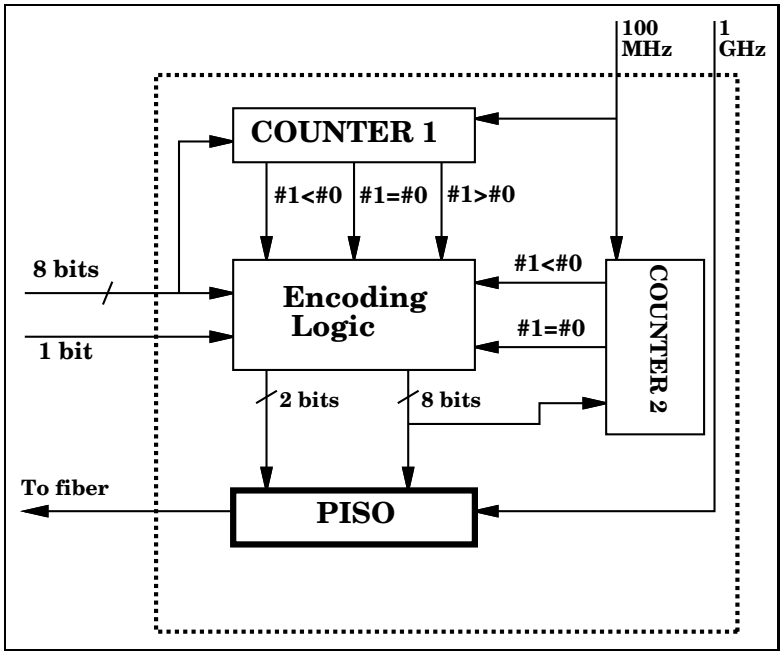


Figure 19.1: Block diagram of our encoder.

Figure 19.1 shows a block diagram of the encoder. Counter 1, counter 2 and the encoding logic

operate on 100 MHz, while the PISO block operates on 1 GHz. Sivasothy has designed the three slowest blocks in $1.5\mu m$ CMOS. They have been simulated on 100 MHz and found to operate. The total area of this block is $3.473mm \times 1.952mm$. The block consists of 2291 transistors. Figure 19.2 shows the layout of the encoder.

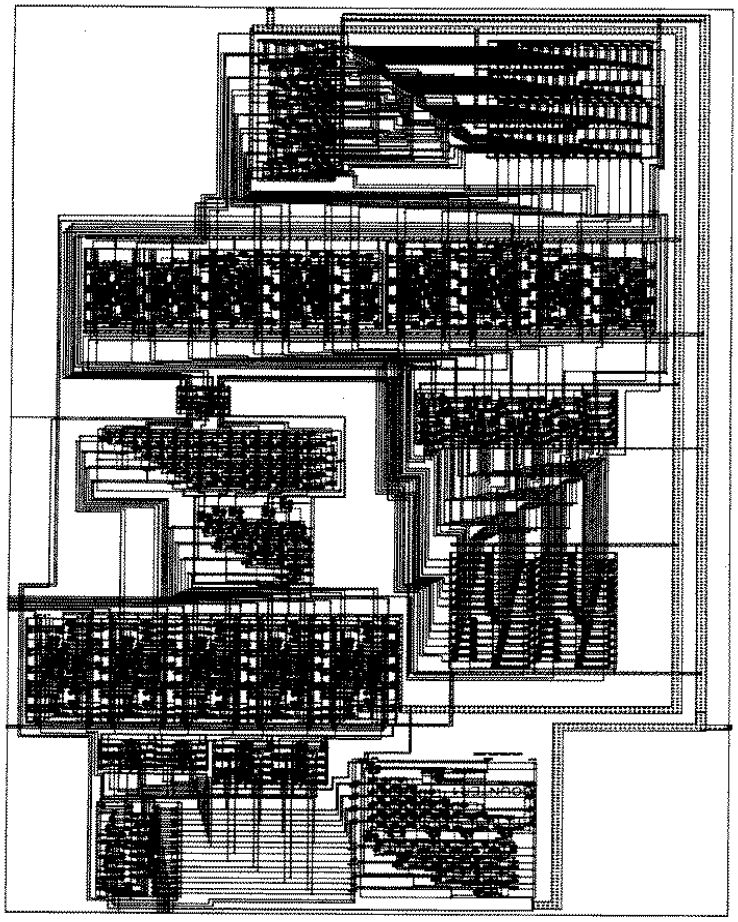


Figure 19.2: Layout of the encoder of Sivasothy.

19.0.2 The decoder implementation by Sivasothy.

Figure 19.3 shows a block diagram of the decoder. Counter 3, the decoding logic, and the latch at the bottom have been designed in $1.5\mu m$ CMOS. They have been simulated at 80 MHz. Sivasothy writes in his thesis that he expect that the aim of 100 Mbps can be attained by optimizing the transistor sizes. The active area has a size of $1.262mm \times 1.179mm$ and contains 831 transistors. The total area with pads is $2.398mm \times 2.305mm$. Figure 19.4 shows the layout of Sivasothy's decoder.

19.0.3 The non-implemented parts.

The designs of Sivasothy cover only the digital parts with the lowest pressure on speed. The parts not designed are the high speed PISO (Parallel In Serial Out) of the transmitter, the SIPO (Serial

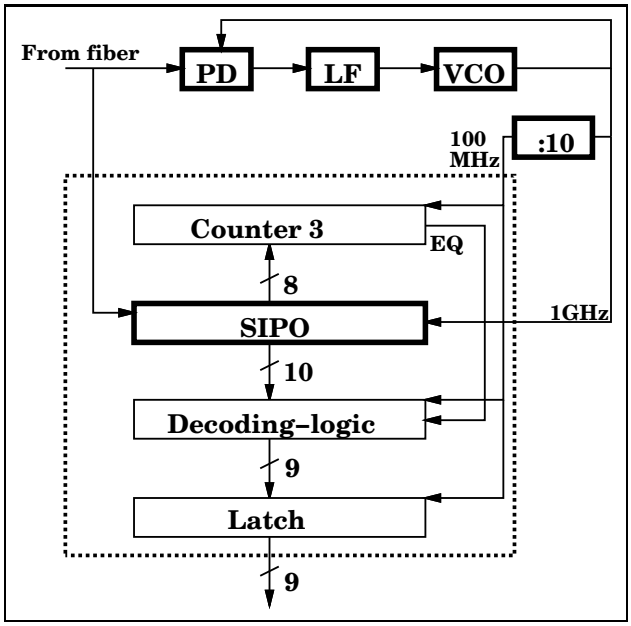


Figure 19.3: Block diagram of decoder proposed for SWIPP.

In Parallel Out) of the receiver and the analog and digital components of the Phase-Locked Loop. Smaller line widths and newer logic families make it possible to use CMOS at progressively higher clock speeds. One of the more critical parts is the Oscillator. A popular CMOS example is given in figure 19.5. This oscillator has been reported to operate on at least 200 MHz [65].

For clock rates approaching 1.0 Gbps it is still necessary to use bipolar transistors or Gallium Arsenide. The BiCMOS technology offers a possibility for bipolar transistors at high speed and in analog parts, and CMOS transistors in the more low-speed digital parts.

19.1 Commercially available chip sets

In this section we describe some commercially available chip sets which can be used with SWIPP.

19.1.1 HDMP-1012/1014 from Hewlett-Packard.

HP offers a pair of silicon circuits [80] for point-to-point connections with clock rates up to 1.25 Gbps. The two M-Quad 80 packages function as virtual ribbon cables that carry parallel data through a single fiber-optic cable for distances up to 10 kilometers. They are implemented in emitter-coupled logic in a silicon bipolar process with a transition frequency (f(t)) as high as 25 GHz. The pair of circuits may be used to transfer parallel data between IEEE HIPPI nodes or between IEEE SCI-FI (Scalable Coherent Interface-Fiber Interface). The selectable data range is from 100 to 1,250 Mbps. Parallel ECL bus options enable transmission of data words in 16-, 17- , 20- or 21-bit widths.

The price of each chip is NOK 509 corresponding to USD 77.

The gigabit-link chip also uses the CIMT code scheme but with input words of either 16 or 20 bits. An additional 17th or 21st bit can serve as a flag bit and is encoded into a control nibble.

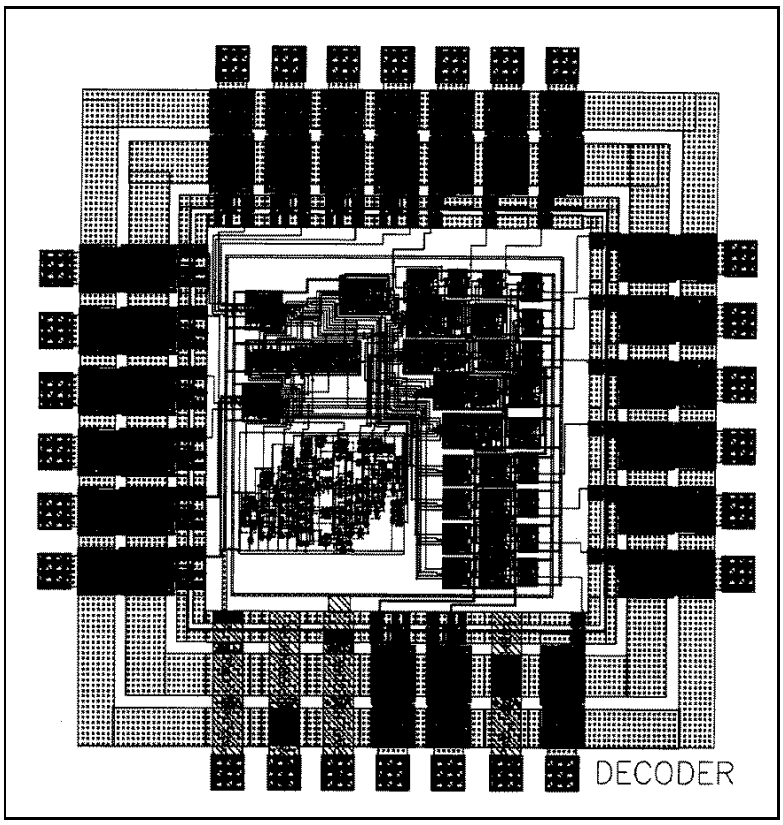


Figure 19.4: Layout of the decoder of Sivasothy.

The control nibble indicates whether the data are true or inverted. This ability to invert data words allows the gigabit-link chip set to maintain DC balance regardless of the data pattern transmitted. Disparity between ones and zeros is accumulated and data words are inverted as necessary. The control nibble also contains a known transition that allows the receiver chip to frame and retime the incoming data. With the CIMT coding the gigabit-link chip set establishes phase lock, frame lock and frequency lock with less than 1,000 logic gates.

19.1.2 Hot Rod 800M

Hot Rod 800M is a transmitter and receiver pair for point to point connection. They are implemented in the TriQuint $1\mu m$ GaAs process. The bit rate is in the area from 200Mbps to 800Mbps.

40 parallel bits of TTL-level data are read into the transmitter device using a simple asynchronous strobe and acknowledge handshake protocol. The data are encoded into a 50-baud word using standard 4B/5B encoding. The resulting 50-baud code word is changed serially across a positive ECL differential interface (across either coaxial or fiber optic media) to the Hot Rod receiver. The receiver recovers both clock and data information from differential input signals. The data are decoded and the 40-bit word is reconstructed for output on a TTL bus. The appearance of new data on the output bus is indicated by a rising edge on the receiver strobe signal. All internal clocking and control functions are transparent to the user.

The Hot Rod chip set is packaged in a 68-pin PLCC.

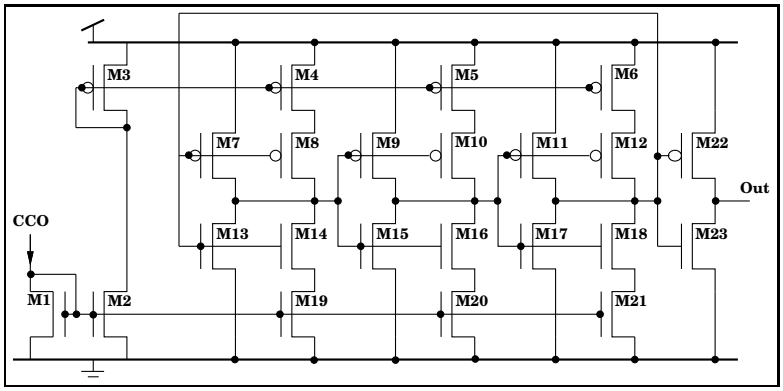


Figure 19.5: A High-Speed Current-Controlled Oscillator (CCO).

19.1.3 The TriQuint ESCON 265 and ENDEC 265M

The TriQuint ESCON chip set consisting of a receiver and a transmitter together with the TriQuint ENDEC chip constitute the necessary chips to interface two eight-bit buses with two fibers making one side of a full duplex fiber connection.

The bit rates on the fibers are 265 Mbaud/s. The coding used on the fiber connection is a 8b/10b block code.

The ESCON transmitter and receiver are designed in TriQuint's $0.7\mu m$ GaAs process. They are interfaced with the ENDEC 265M chip through two 10-line buses transmitting in opposite directions.

The ENDEC 265M performs the decoding and encoding of the block symbols. It is implemented in state-of-the-art lowpower CMOS.

Chapter 20

Integration of the SWIPP switch as one circuit.

Except for some of the most high speed parts (above 500 MHz) all parts of the 16×16 *SWIPP switch have been designed and simulated. Layouts have also been made. However, some of the layouts have been made with old design rules. Compared to the state of the art, these layouts may be considered space-wasting. The designers have put variable amount of effort into making the layout compact. This chapter discusses the possibility of integrating a switch on a chip. The discussion is based on the chip area and the transistor count of the designed parts. The discussion also includes power reduction and utilisation of new, smaller, scaled-down technologies.*

20.1 Chip size estimates based on transistor counts

20.1.1 The number of transistors.

A good basis for chip size estimates is the number of transistors used. The simulations of the modules show correct logical behaviour for all modules. Hence, these modules are a good basis for further design. However, some of the modules did not satisfy the clock rate goal. A redesign to achieve higher clock rate will influence only smaller parts of the modules and is believed to give minor changes in the number of transistors.

Table 20.2 shows the number of transistor equivalents for a SWIPP switch when the input buffers are excluded. The high clock rate parts (above 500 MHz) of the Phase-Detector, the Low-Pass filter and the VCO have not been completed for SWIPP. However, based on the literature and our experience from other designs we believe that we can make a good estimate of the number of bipolar transistors required. For simplicity, the calculations on the following pages will be performed based on the number of MOS transistor equivalents. It is not straightforward to find the number of MOS equivalents for a bipolar transistor. This ratio should cover average area and power consumption. Such a ratio has to be based on layouts where varying effort may have been put in reduction of the area and power consumption. Based on the literature examples (table 20.1) and our own examples (table 20.3), the ratio for area is in the range 22 to 2.[1] Similarly, the ratio for power consumption is in the range 48 to 4. The high values are not believed to be

[1] I.e. the average area required for each bipolar transisitor is between 22 and 2 times the average area required for a MOS transistor.

Ref:	[103] PLL	[85] PLL	[4] PLL	[4] PLL+SIPO +PISO	[2] PLL	[27] PLL	[84] PLL
Month published	4/89	2/95	3/95		4/95	4/95	4/96
Serial line rate	8kbps	3Gbps	10Gbps		6-175Mbps	(50-550Mbps)/4	2.5Gbps
P: Power consumption	25mW	25mW	3.2W++	9.1W++	10mW	40mW	15mW
	$3.0\mu m$ CMOS	$0.1\mu m$ CMOS	Bipolar E^2CL		$0.5\mu m$ CMOS	$0.5\mu m$ CMOS	$1.0\mu m$ BiCMOS
Capture range	± 1kHz	± 320MHz					300MHz
N_{Tran} : No. of transistors						6 000M	90B+20R+7C
A: Area	$3.55mm^2$	$0.006mm^2$	$11mm^2$++	$27mm^2$++	$0.52mm^2$	$0.71mm^2$	$0.25mm^2$
Supply Voltage	5V	2.8V	(-)5.2V		3.3V	3.3V	3V
Area/N_{Tran}						$118\mu m^2$	$2777\mu m^2$
P/A (mW/mm^2)	7.0	4166.7	291	338	19.2	56.3	60
P/N_{Tran}						$6.7\mu W$	$166.7\mu W$
A@$1.2\mu m$	$0.57mm^2$	$0.86mm^2$			$3.0mm^2$	$4.1mm^2$	$0.36mm^2$
A@$1.2\mu m$/N_{Tran}						$681.6\mu m^2$	$4000\mu m^2$

Table 20.1: The table shows some example PLLs (Phase-Locked-Loops) from the literature. The one in the second column is old, designed in a technology with $3.0\mu m$ gate length. The third column shows the values for a PLL designed in an advanced technology not commonly available. The literature example shown in column four contains a description of PLL, SIPO (Serial-In Parallel-Out) and PISO (Parallel-In Serial-Out). The values for the PLL alone are shown in the left part of the column, while the values for the complete logic are shown in the right part. This circuitry has the highest bit rate, but also the highest power consumption and requires the largest area. The fifth and sixth column show the values for two PLLs designed in $0.5\mu m$ gate length CMOS technologies. The main difference between them is in serial bit rate. The last column shows the values for a bipolar PLL designed in a $1.0\mu m$ BiCMOS technology. The architecture described in this column is the one believed to be best suited for SWIPP. Thus, these values are chosen as example values for SWIPP. In the two rows at the bottom, the circuit areas have been scaled to an $1.2\mu m$-reference technology. The area scaling factor is $(1.2/x)^2$, where x is the gate length of the initial technology.

		Components	MOS equivalent	Total for module	No. of modules	Total
CSU		40 672 MOS		40 672	1	40 672
IP	without FIFO	2 142 MOS		2 142	16	34 272
OP		2 006 MOS		2 006	16	32 096
OM-T	Encoder	2 291 MOS				
	10bit PISO	180 BJT + 100 R	*1 800*	4 091	16	65 456
OM-R	Decoder	831 MOS				
	10bit SIPO	180 BJT + 100 R	*1 800*			
	Phase Detector	30 BJT	*300*			
	Low Pass Filter	30 BJT	*300*			
	VCO	30 BJT	*300*			
	Down counter :10	72BJT+ 40 R	*720*	4251	16	68 016
						240 512

Table 20.2: The number of transistor equivalents for the SWIPP switch without main buffers. A bipolar transistor is sat equivalent to 10 MOS transistors.

representative. The author has chosen a ratio of 10 and believes this to be an exaggeration of the power consumption and area consumption of bipolar transistors.

20.1.2 Module areas when scaled to a $1.2\mu m$ technology.

Table 20.3 shows some of the parameters for the designed modules related to area consumption. The modules have been designed for different technologies (minimum gate lengths) and with different design rules. Scaling into one common technology is not always straightforward: When a $1.2\mu m$ layout is considered scaled to a $0.6\mu m$ technology, it is rarely so that all layout lengths can be reduced to a half. Typically, this problem is greater when comparing technologies from different silicon foundries. The silicon foundries usually emphasise correct scaling of the layout design rules having the largest influence on chip area: width and separation of contact holes (via-contacts included) and metal. Thus, after some redesign a scaling from a $1.2\mu m$ technology to a $0.6\mu m$ technology can give a reduction of lengths to a half and a reduction of area to a quarter.

On the right side of 20.3 we make a simplification. We say that a design scaled to a $1.2\mu m$ technology is $(1.2/1.5)^2$ times the area in the $1.5\mu m$ technology. In column 8 of table 20.3 we find the new, scaled areas. By adding up the required elements from column 8, except the bipolar parts and the FIFOs, we find a total area consumption of $138.5mm^2$. If we include the bipolar transistors with the MOS equivalents from table 20.2 and give each of the transistor equivalents a size of $350\mu m^2$, we have an area consumption of $166.6mm^2$. This is a very large design leaving no space for FIFOs.

Modules		Original					After scaling to $1.2\mu m$ technology		
	Designer: Ref.	Sim. clock rate	Expected clock rate	Design rules	Size in mm^2	Number of transistors	New size in mm^2	Area pr. transistor μm^2	Bit size in μm^2
RAM core cell				$1.2\mu m$	$400\mu m^2$	6	$400\mu m^2$	67	
RAM eff. cell				$1.2\mu m$	$964\mu m^2$	12	$964\mu m^2$	80	964
High speed FIFO:FC	JMØ:App.G	100MHz	200MHz	λ-rules		1654	0.293	177	4069
OP:FC	JMØ:App.B	100MHz	200MHz	λ-rules		2006	0.396	197	
IP-pipeline:FC	JMØ:App.B	100MHz	200MHz	λ-rules		2142	0.470	219	
IP Async FIFO:	PTR:[86]	100MHz	100MHz	$1.5\mu m$	9.5	19000	6.080	320	10083
OM-T:FC	MRS:[94]	100MHz	100MHz	$1.5\mu m$	6.8	2291	4.339	1894	
OM-R:FC	MRS:[94]	80MHz	80MHz	$1.5\mu m$	1.5	831	0.952	1146	
CSU 4x4 CMOS:FC	JEK:[63]	80MHz	80MHz	$1.5\mu m$	6.0	6000	3.840	640	
CSU 16x16:SC	JMØ:App.D	100MHz	100MHz	$1.2\mu m$	40.0	57376	40.000	697	
CSU 4x4 ECL:FC	MM:[70]	100MHz	100MHz	$2.0\mu m$	7.5	2004	2.800	1394	
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)

Table 20.3: The table shows some of the parameters for the designed SWIPP modules. Column 3 is the clock rate for which the design has been verified. Column 4 is the unconfirmed expected clock rate. The other columns should be self-explaining.

20.1.3 Average area per transistor.

Column 10 in table 20.3 gives some interesting information about the possibility of, and the designer effort in making the layout dense. The whole VLSI community is interested in dense RAMs. In our table, the RAM cells have the highest density, with an average area of $80\mu m^2$ per transistor. The next three modules, designed by this author (the high speed FIFO, the OP and the IP-pipeline) require more than twice this area per transistor. Much effort was put in making them dense. When we compare with the other designs, we see that the effort did pay. The asynchronous FIFO designed by one of the master students of the author, requires $320\mu m^2$ per transistor. This is a full custom design. The asynchronous FIFO was initially generated with standard cell tools. The designer writes in his thesis ([86] p.94) that the full custom size was 40 % of the standard cell design. This implies that the average area per transistor in the standard cell design was $800\mu m^2$. The next two circuits were designed by another master student supervised by the author. The encoder and decoder of the parallel-serial converters have been designed with very much space per transistor. Thus, these designs clearly have a potential for increased density. Actually, their area dominate so much of the total switch area (more than 50%) that their layout has to be redesigned. The 4x4 CMOS CSU, designed by a third master student, is partially a full custom and partially a standard cell design. However, the area per transistor is in the range of the standard cell designs. The 16x16 CSU is a standard cell design made by this author. It was generated from a standard cell library by an auto-place-and-route tool. No additional compacting was done by tool or designer. Since the CSUs, and especially the 16x16 CSU, are heavy on wiring, its achieved area per transistor ($697\mu m^2$) should be regarded as an absolute upper limit on average area per transistor. The last design, the 4x4 ECL CSU, is different from the others: it is designed with bipolar transistors. The average area per bipolar transistor, $1394\mu m^2$ includes wiring, the bipolar transistor and a part of the resistors.

Based on the discussion above and the numbers in table 20.3 we will for the following discussion define set a standard. Standard Cell (SC) design generated by Place-And-Route tools should

Original						At 100MHz		
		Technology	Clock rate	Power	No. of transistors		Total power	Power pr. transistor
CSU 4x4 CMOS	p. 170	$1.5\mu m$	100 MHz	250 mW	6 000	Sim.	250 mW	$42\mu W$
						Calc.	293 mW	$49\mu W$
Async. FIFO	p. 187	$1.5\mu m$	50 MHz	0.1-0.4W	19 000	Sim.	0.2-0.8W	$11-42\mu W$
						Calc.	920mW	$48\mu W$
CSU 4x4 ECL	p. 170	$2.0\mu m$	100 MHz	640 mW	$320\mu W$	Sim	640 mW	$320\mu W$

Table 20.4: Estimated and simulated power consumption.

have a density so that the average area per transistor is less than $700\mu m^2$. For an $x-\mu m$ technology (i.e. a technology with a minimum gate length of $x-\mu m$) the area should be less than $(x/1.2)^2 \cdot 700\mu m^2$. Full custom design should have a density giving an average area per transistor less than $350\mu m^2$ or less than $(x/1.2)^2 \cdot 350\mu m^2$ in the general case for a $x\mu m$ technology. For most technologies the silicon foundries are offering there are available RAM generators giving the highest density. Thus, the higher density expected for RAM modules should be managed by the RAM generators. The numbers above should be obtained easily. With some effort it should be possible to come significantly below these numbers.

20.1.4 Average power consumption per transistor.

For the calculations in the following pages we need an estimate of the power consumption for different technologies and different clock rates. A calculated power estimate can be done with the following equation from [6] eq. 9.28 p. 441:

$$P_{Tran} = \frac{1}{2} f_{clk} f_d V dd^2 C_{Load} \tag{20.1}$$

f_{clk} is the clock frequency. Vdd is the supply voltage. C_{load} is the estimated average capacitive load for each transistor. The estimated load for a design in $1.2\mu m$ technology has been sat to $100fF$. To scale the capacitive load to other technologies with other minimum transistor lengths the scaleing factor $(x/1.2)^2$ has been used. Here x is the minimum transistor length of the destination technology. In CMOS only gates changing logical values consume power. The factor f_d represents the proportion of transistors changing state. C_{Load} and f_d is the factors most difficult to estimate. In the following we will keep C_{Load} as defined above, and let linear errors in C_{Load} be included into the f_d factor. In the following we will discuss a good value for f_d.

Table 20.4 shows simulated and estimated power consumption for some circuits. The simulated power consumption has been found by the use of analogue simulators. The estimated power consumption has been calculated with a $f_d = 1/4$. For the 4×4 CMOS CSU we have $f_d = 1/4.7$. For the asynchronous FIFO, f_d is in the range 1/4.6 to 1/18.4. No special effort has been put in reducing the power consumption of these circuits. A smaller power consumption may be expected from the asynchronous FIFO when idle. By calculating on one of the CMOS PLLs ([27]) in table 20.1[2], we find a f_d of 1/8. Thus, choosing a f_d equal to 1/4 should give an upper estimate of power consumption.

Estimating power consumption in this way may look inaccurate, but processed circuits also have variations. By simulating on minimum and maximum model parameters from silicon foundries,

[2]This is the only CMOS paper referred to in this table containing both the number of transistors and the power consumption.

we find values between -20% and 180% of the typical values.[3] Thus, it is not very fruitful to try to reach a better accuracy of estimates than what the silicon foundries can achieve themselves.

20.1.5 Model for chip temperature estimates.

Too high chip temperatures reduces both the lifetime of integrated circuits and their electrical behaviour. These effects increase continuously with temperature. Thus, the acceptable upper temperature limit depends on the requirements on the lifetime and electrical behaviour. A temperature often chosen as the upper design limit is 85^OC.

The chip temperature can be expressed by the following equation:

$$T_j = \theta P + T_a \tag{20.2}$$

Here T_j is the chip temperature and T_a is the ambient temperature. The ambient medium is a strong "temperature source" that is not influenced by the power consumption and the temperature of the inspected chip.[4] The ambient medium may be air and/or a large substrate. P is the power consumption of the chip while θ is the thermal resistance between the chip and the ambient material. From the equation we see that to attain a low chip temperature we have three possibilities: reducing the power consumption, reducing the thermal resistance and reducing the ambient temperature. In the following we will focus on the thermal resistance. In the examples on the following pages we assume an ambient temperature of 25^OC.

The most impotant choices influencing on the thermal resistance are the choices of packaging, substrate and heat sink. For the estimates on the following pages the choices have been taken according to the implementation proposed by this author: flip-chip mounting. The author assumes the thermal resistance between the chip and the substrate to be dominating. For simplicity the thermal resistance between the substrate and air/the surroundings are assumed to be zero, i.e. the substrate has a temperature of 25^OC. This is obviously not correct. The approximation to this assumption depends on the efficiency of the substrate heat sink.

The author has measured the thermal resistance of six flip-chip-mounted circuits. The variation in thermal resistance was 32%. One reason for the variation was the different number of bump connections. When the thermal conductance for each bump was found the variation was reduced to 19%. The thermal conductance of the bumps used in this case was measured to be $0.001W/^OC$.

Assumptions for the temperature estimates on the following pages:
- substrate temperature of 25^OC,
- flip-chip-mounted circuits with approximately 200 bumps,
- a thermal conductance per bump of $0.001W/^OC$.

20.1.6 Space trade-off between improved arbitration and buffer.

Chapter *13 The arbitration logic of the CSU* discusses the performance of different arbitration algorithms. From the chapter and also from table 16.1, it is clear that fair arbitration requires

[3] This is between different circuits. Inside one circuit a far better matching can be achieved.

[4] This has similarities with the electrical decoupling of AC signals to strong power lines. The power lines have "easy" access to charges that compensate for the relatively smaller fluctuations of individual signal lines. Thus the potentials of the power lines are maintained.

	Technologies			
	$1.2\mu m$	$0.8\mu m$	$0.5\mu m$	$0.35\mu m$
FC:	$83mm^2$ 2.2kB FIFO 14W 82^OC	$36.9mm^2$ 18.4kB FIFO 28W 140^OC	$14.4mm^2$ 63.9kB FIFO 35W 170^OC	$7.1mm^2$ 141.7kB FIFO 37W 178^OC
SC:	$166mm^2$	$74mm^2$ 7.6kB FIFO 14W 83^OC	$29mm^2$ 53.1kB FIFO 29W 140^OC	$14mm^2$ 130.9kB FIFO 34W 170^OC

Table 20.5: Switch size without buffer, buffer size in bytes and power consumption. The values are given for a 5Volt power supply. The power consumption is based on a 100 MHz clock rate and a capacitive load on each gate of $(x^2/1.2^2)100fF$ *for an* $x - \mu m$ *technology. FC is Full Custom design while SC is Standard Cell design from a place-and-route tool. The FIFOsize and power consumption are found for a total circuit size of* $100mm^2$*. If the area required for the circuitry without buffer is above this size, the other numbers are not given.*

more transistors and more space than an unfair arbiter. Thus, a natural question is how much space can be saved for buffers when an unfair arbiter is chosen, compared to a fair one. Since arbiter size and buffer space are scaling similarly, it is even more interesting to figure out how much buffer in bits corresponds to the differences in the arbiters.

According to table 16.1 the difference between the fast unfair and the fast fair is 7 100 transistors when the smallest latch is used. With standard cell transistors this corresponds to 640 bytes, when we convert buffer space according to table 20.3. If the largest latch is used in the arbiter, the difference is 15 600 transistors which corresponds to 1.5 kbytes. Since these numbers are based on two equally scaled areas, the size of the buffer equivalents will not change with scaling. Thus, when further scaling of technology gives larger buffers on-chip, these buffer equivalents will contribute less significantly to the total buffer amount and a fair arbiter should be preferred.

20.2 Example estimates: Size, buffer and power for different scaling and different voltage.

Table 20.5 shows that for a $1.2\mu m$ technology the design has to be full custom (FC). With an average area per transistor of $350\mu m^2$ the switch (RAM not included) will occupy approximately $83mm^2$. Assuming a maximum chip size of $100mm^2$. The remaining $17mm^2$ of the chip area is left for FIFO RAM. The total RAM is 2.2kbyte. The upper power consumption is estimated to 14W. For scaled-down technologies, the buffer size may increase and thus the power consumption will increase as shown in table 20.5.

To reduce the power consumption, the voltage supply can be reduced. A voltage supply reduction from 5 Volt to 3.3 Volt increases the circuit delays with a factor of 1.6 (Table 4.3). To keep the same bandwidth the bus width has to be increased with the same factor. If we regard the number of transistors to be proportional to the bus width we have the number of transistors as given in

	Technologies			
	$1.2\mu m$	$0.8\mu m$	$0.5\mu m$	$0.35\mu m$
FC:	$133mm^2$	$59.3mm^2$ 11.9kB FIFO 9W 63^OC	$23.2mm^2$ 57.4kB FIFO 14W 83^OC	$11.4mm^2$ 135.1kB FIFO 16W 89^OC
SC:	$267mm^2$	$119mm^2$	$46.4mm^2$ 40.1kB FIFO 10W 66^OC	$14mm^2$ 117.8kB FIFO 14W 81^OC

Table 20.6: Same as previous table, but with a power supply of 3.3Volt and increased channel width to give the same bandwidth.

	Technologies			
	$1.2\mu m$	$0.8\mu m$	$0.5\mu m$	$0.35\mu m$
FC:	$186mm^2$	$83mm^2$ 4.9kB FIFO 4W 40^OC	$32mm^2$ 50.4kB FIFO 7W 55^OC	$15.9mm^2$ 128.2kB FIFO 9W 60^OC
SC:	$374mm^2$	$166mm^2$	$64.9mm^2$ 26.2kB FIFO 4W 42^OC	$31.8mm^2$ 104.2kB FIFO 7W 54^OC

Table 20.7: Same as the two previous tables, but with a power supply of 2.5Volt and increased channel width to give the same bandwidth.

Voltage	Bus scaling factor	No. of transistors.
5.0V	1.0	240 512
3.3V	1.6	384 820
2.5V	2.3	553 178
1.5V	4.5	1 082 304
1.0V	9.0	2 164 608

Table 20.8: The number of transistors in the SWIPP switch when bus width compensates for reduced circuitry speed. The FIFOs are not included in the switch area.

	Technologies			
	$1.2\mu m$	$0.8\mu m$	$0.5\mu m$	$0.35\mu m$
FC:	$373mm^2$	$166mm^2$	$64.9mm^2$ 26.2kB FIFO 2W 32^OC	$31.8mm^2$ 104.0kB FIFO 3W 40^OC
SC:	$747mm^2$	$332mm^2$	$124mm^2$	$63.6mm^2$ 55.5kB FIFO 2W 31^OC

Table 20.9: Same as the previous, but power supply of 1.5Volt and increased channel width to compensate for the lower clock rate.

table 20.8. The tables in 20.6, 20.7 and 20.9 show chip parameters for different voltages where the number of transistors has been scaled as described above.

Table 20.6 shows that the $1.2\mu m$ technology can not be used for a power of 3.3V. A circuit designed in $0.8\mu m$ for 3.3V requires full custom design. Smaller scaling ($0.5\mu m$ and $0.35\mu m$) allows both full custom and auto-routed standard cells.

Table 20.7 shows the same parameters for a circuit designed for 2.5V power. Similarly to the circuit designed for 3.3V, design in $1.2\mu m$ and standard cell $0.8\mu m$ requires too much space for a $100mm^2$ circuit. The power consumption with 2.5V is half of the power consumption for 3.3V.

Table 20.9 shows the values for a 1.5Volt voltage supply. To implement the complete switch on one circuit the design has to be done in $0.35\mu m$ or in $0.5\mu m$ full custom.

20.3 Conclusion

For an integration on one $100mm^2$ circuit in a $1.2\mu m$ technology, the layout of several modules has to be made more compact. The power consumption is approximately 14W. In a $0.8\mu m$ technology, the complete design may be done with auto-place-and-route tools (fulfilling the standard cell density defined above). The transistor structures on the schematics may be kept. Due to different

transistor sizes and capacitive loads, the structures have to be resimulated. This author finds no reason why the transistor structures should not work for a smaller technology. The lay-out of some of the least dense modules has to be redesigned. They may be described as standard cells for an auto-place-and-route tool or if the time is available for full custom.

For the smallest scaling the power consumption increases significantly. Since low power consumption is a goal, redesign for a lower voltage should be started. As discussed several times in this thesis (table 4.3), a reduction of power voltage reduces the clock rate. However, the same bandwidth may be kept through increased bus width. The new power consumption will be smaller. The logical behaviour of the designed modules may be kept. This requires a change of the transistor schematics. During the redesign, the structure of the cells should be inspected for a general reduction of voltage. The first step may be to $3.3V$ or $2.5V$ but it can almost be guaranteed that a need for $1.5V$ and $1.0V$ will follow. Such a redesign requires transistor schematic changes, resimulation and new layout.

PART 6

CONCLUSION

Chapter 21

Summary.

This summary contains the main points made in this thesis.

21.0.1 The design goals.

The design goals for the SWIPP network were as follows:
a) high network performance, (4.1.1 of this thesis)
b) minimum interaction with host processor, (4.2)
c) expansion with small low-cost units should be easy, (4.4)
d) good scalability, (4.5)
e) large connectivity, (4.6)
f) large traffic flexibility, (4.7)
g) low power consumption, (4.8)
h) small physical size, (4.9)
i) reduction or avoidance of major fault situations like deadlock, (4.10)
j) attaining the maximum knowledge about the expected network traffic of the applications. (4.11)

21.1 The arguments for the SWIPP architecture.

21.1.1 Main performance bottlenecks.

Based on literature study and the author's own analysis and simulations, it is clear that the bottlenecks for high performance are :
1) complicated communication protocols in the end nodes,
2) too long involvement of the computing engine in communication protocols,
3) low network bandwidth,
4) network topologies and switch architectures giving high contention, and
5) long propagation time through the network.

The bottleneck elements are listed with the main contributors to the worst case performance degradation first.

21.1.2 Protocol engine.

To reduce the Computing engine involvement in communication tasks (bottleneck 2) two strategies should be followed: as much as possible of communication tasks should be off-loaded to hardware and processor in the Protocol Engine. Secondly, data should be read and written as fast as possible directly to/from the Computing Engine, preferably with a DMA operating on the maximum speed of the Computing Engine memory. This will make the Computing Engine available for other application tasks during the communication.

To reduce latency (bottleneck 1 and 5), data should be forwarded directly between network and host memory (both directions) without temporary storage in Protocol Engine memory.

Faster network protocols (bottleneck 1) have been and will be developed further. A literature study has been presented (chapter 6 and section 4.1.1). Faster protocols require mainly a change of Protocol Engine hardware, Protocol Engine software and Computing Engine software. The network core may contribute to simplifying network protocols through more reliable communication.

SWIPP has a unique packet format. The Protocol Engine may pack and unpack most other packet formats inside a SWIPP packet. Both IP packets and ATM cells may be carried as payload in the SWIPP packets. The SWIPP Protocol Engine simply reads the destination addresses from the packets, performs a look-up in a table of SWIPP addresses and forwards the packet with the SWIPP address through the network.

21.1.3 Reliable connections: buffering, flow control and deadlock-free routing.

Low error rate and low packet dismiss rate are important to increase the efficient utilisation of bandwidth, to reduce the packet propagation time and to support simplification of communication protocols.

Networks without switch buffers require retransmission when collisions occur. Retransmission is obviously bad utilisation of bandwidth. A result is increased probability for new contention on the network segments passed twice. Thus buffers may only be avoided on small networks with low load and where reliable delivery is not very important.

Flow control secures packet loss when buffer capacities are exceeded. Networks without flow control require good knowledge about traffic statistics. The networks are sensitive to variations in traffic pattern and the dismiss rate may increase significantly if the traffic pattern changes too much. The buffer requirement is large, several times the messages transmitted[1]. Naturally, this is a problem for transmission of large data files. Networks without flow control typically require a predictable traffic pattern and small data units, and can not be very dependent on reliable delivery. While some switches, especially ATM switches, have been designed without flow control, the trend is that high performance data networks have flow control.

Deadlock (circular dependence of resources) may result in total blockage with no propagation in a large part of a network. Deadlock may be avoided through choice of topology or logical routing algorithms. The disadvantage is that some of the bandwidth is restricted. Due to the serious damage from deadlock, deadlock-free routing strategies should always be used.

[1] Dividing a message into packets does not change this fact. It is the size of the data group transmitted together which gives the queueing length.

When the probabilities of packet dismissal and errors are small, packet parts can be transmitted as soon as they are received and decoded ("wormhole"-routing). This is important to reduce the packet propagation time.

SWIPP has buffers with flow control and assumes a dead-lock-free routing algorithm.

21.1.4 Star switches.

The SWIPP switches with 16 full duplex channels are believed to be a flexible basic element for a number of regular and irregular topologies. A few switches can be used to establish a minimum network with fair performance. More elements can be added to increase bandwidth between switches or to reduce latency through direct connections. More switches may also be added to give topologies with less blockage. The switches may be connected in regular structures to mimic rings, meshes, n-dimensional networks, k-ary n-cubes etc. Alternatively, the switches may be connected in an irregular structure for a specific traffic pattern.

21.1.5 Input buffered switches.

The SWIPP switch has an architecture with strict FIFO queues at the input channels. The advantage of this structure is that variable packet lengths will not complicate the buffer structure. Thus, as chosen for SWIPP, the hardware may allow variable packet lengths with no maximum length. Thus, it is possible to forward packets of any size without segmentation. This gives an increased utilisation of the bandwidth, since only one packet head is required.

A disadvantage of this switch architecture is the reduced bandwidth utilisation, due to Head-Of-Line blockage (HOL). This occurs in all systems with a strict FIFO queue, where a negative flow control signal can stop the first packet. Packets for unoccupied channels are not allowed to pass, and this reduces the traffic level at which saturation occurs (the saturation level). HOL may reduce the switch utilisation down to 56 % when all input channels choose between all output channels with equal probability. If they transmit to a limited set of output channels with higher probability, or if the input channels have private preferences for output channels, the differences in performance are smaller. HOL blockage also occurs in centrally buffered and output buffered switches with strict FIFO buffers and flow control.

Reduction of the HOL effect to increase the saturation level can mainly be obtained in two ways: by using a switch architecture that allows bypass, or by connecting the switches in a topology that reduces the HOL effect. Examples of bypass architectures are "FIFO"s with multiple outputs and buffer banks with buffer segments reserved for dedicated connections or groups of connections. These architectures are more complicated to design, use more space for control logic and require fixed packet sizes. They also have a higher minimum requirement on buffer size.

The acceptance of HOL blockage in SWIPP is defended by the low-cost design goal. If the cost of a SWIPP switch is less than half the cost of switches with higher saturation levels, adding more SWIPP switches may give equal or better performance at a lower total cost.

21.1.6 Short time to establish connections: Source routing and distributed arbitration.

With source routing the entire packet route is decided by the transmitting Protocol Engine. This simplifies the switch architecture, since no table look-up is required. Neither is circuitry or protocols for initiation and up-dating of the address table needed. New connections can be established faster and independently of other connections.

On-chip the switches have a private arbitration block at each output. This supports shorter time to establish a connection, giving reduced network latency.

An unfair arbiter can be implemented with less transistors and less space than the fair alternatives. A fair arbiter will give more equal treatment of the requesting input channels for the highest load levels. For lower load levels the difference will be insignificant. As further scaling of technology gives larger buffers on-chip, the buffer equivalent of the arbiter difference will contribute less significantly to the total buffer amount, and will no longer be important for the choice of arbiter algorithm. Since fair local arbitration favours the closest hosts, unfair local arbitration may be used to achieve a more fair global treatment of traffic. To do this, switches have to be connected with the highest priority channels to other switches while local hosts have lower priority.

21.1.7 Network links: parallel or serial, twisted pair, coaxial cable and optical fibre.

The choice of link medium depends on the physical distance between the net nodes and on other circumstances like space, radiation etc. For short distances parallel twisted pair cables may give shorter propagation time. For longer distances optical fibre have a higher bandwidth, lose less of the signal strength and require less space, but they are more expensive. For medium distances, coaxial cable may be a suitable compromise both in performance and price.

When standard switches are being designed, circuitry for transfer between "low speed" parallel and high speed serial[2] code format should be included on all links. If the distance between the net nodes is small and a direct connection would be faster, the parallel-serial converters can be bypassed. On the other hand, the "unnecessary" delay at short distances is insignificant relative to the other contributors to the total network latency.

21.1.8 High bandwidth.

High bandwidth is one of the most important premises to achieve high performance. A doubling of bandwidth reduces the time to forward a packet on a channel to a half. It also reduces the average queuing time to less than a half. High bandwidth should be attained through high serial and parallel capacity. The bandwidth utilisation should also be high. In SWIPP this is supported through the flow control system, which occupies an absolute minimum of the bandwidth. It is also supported through the variable packet format where no additional headers or dummy data have to be inserted. However, SWIPP is suffering from the HOL saturation. The utilisation level may be increased towards 100% through choice of routing strategy and topology.

In network architectures like Telegraphos (chapter 6), due to packet headers and intensive ex-

[2]For both coaxial cable and optical fibre.

change of tokens the bandwidth utilisation can never reach 100 %. For the version of Telegraphos I presented in 1994, the absolute maximum utilisation is 80 %. For the Telegraphos II version the maximum utilisation is 90 %, while with full ATM packet length, the maximum utilisation will be 93 %. The dummy data that fill up the last packet also reduce the utilisation. When the average message length is only a few packet lengths, the bandwidth utilisation shrinks to only 60 %.

21.2 Switch implementation.

21.2.1 Technology: Full custom BiCMOS.

BiCMOS is the most attractive technology for a design of the SWIPP switch as an integrated circuit. BiCMOS allows CMOS logic, BiCMOS logic and ECL logic on the same circuit. CMOS logic is the logic with the highest density of transistors and logical functions per area. Newer CMOS and BiCMOS technologies have demonstrated clock rates of several hundred MHz. Clock rates above 1 GHz are expected in a few years. CMOS and BiCMOS are also the technologies with the potential for the lowest power consumption. Fixed bandwidth can be kept by reducing serial capacity and increasing parallel capacity to reduce the total power consumption. Thus, CMOS logic is suitable for 100 % of the memory and more than 60 % of the remaining circuitry. The part where BiCMOS or ECL logic is required, is the high speed logic inside the parallel - serial converters at the link inputs and outputs. The development of the technologies is expected to support the considerations and decisions made above.

Compared to other design solutions, full-custom design offers the possibility of better utilisation of the technology to give higher clock rate, higher bandwidth, lower power consumption and higher density.

21.2.2 SWIPP switch integrated as one circuit.

In chapter 20 an integration of the SWIPP as one circuit is discussed. Some layouts should be redesigned to make them more compact. To get the entire switch on one chip a $0.8\mu m$ or smaller technology should be used. For these smaller sizes the power dissipation density becomes significant and power reduction alternatives have to be considered.

In table 21.1 one technology is chosen, and values for different voltages are given. We see that for decreasing voltage supply the number of transistors for the switch part increases, the buffer decreases and the power consumption decreases.

Silicon has a high sensitivity for wavelengths in the area around $800nm$. Thus, if light of this wavelength is chosen the optical fibre may terminate directly on the BiCMOS switch circuit. The light-emitting semiconductors have to be separate devices. Silicon is an inefficient light emitter. Light-emitting semiconductors require special processing steps not available in standard CMOS and BiCMOS circuit technologies.

21.2.3 Pin number.

Different solutions requiring different numbers of pins have been discussed in this thesis. The solutions may be divided into two types: one type where the complete switch is integrated on

	5V	3.3V	2.2V	1.5V
No. of tran.	240512	384820	553178	1082304
FC:	$14.4mm^2$	$23.2mm^2$	$32mm^2$	$64.9mm^2$
	63.9kB	57.4kB	50.4kB	26.2kB
	35W	14W	7W	2W
	168^OC	83^OC	55^OC	32^OC
SC:	$29mm^2$	$46.4mm^2$	$64.9mm^2$	$124mm^2$
	53.1kB	40.1kB	26.2kB	
	29W	10W	4W	
	145^OC	66^OC	42^OC	

Table 21.1: This table shows some estimated values for a complete 16×16 *switch implemented on one circuit. The estimate has been performed with a 100 MHz clock rate for a* $0.5\mu m$ *technology with different supply voltages. The estimated number of transistors are the number of CMOS transistor equivalents of the switch without the input buffers. The area given is the estimated area for the same part. The table also shows the estimated buffer size on the remaining area presuming a total chip area of* $100mm^2$*. The transistor density for FC (Full Custom), SC (Standard Cell) and the power consumption model are given in the previous chapter*

one circuit and one type where the CSU, the Input and Output Ports and the Optical Modules are integrated on separate circuits.

A one-chip solution requires redesign of circuit layout. If the photo detectors are integrated, the circuit requires only 16 output lines operating with a bit rate of 1 Gbps per line. If the photo detectors are not integrated, 32 lines operating on 1 Gbps each, are required. Due to short distance between circuits or for other reasons, it may preferable to have the possibility of bypassing the optical modules. With the 9-line-wide buses (100 Mbps per. line) discussed in this thesis a total of 320 data pins are required. This number includes 32 lines with a line bandwidth of 1 Gbps and 288 lines with a line bandwidth of 100 Mbps. This number is high and a better solution may be to double the bit rate of the slowest lines to halve the number of pins required. This would give a total of 192 pins (32 at 1 Gbps and 160 at 200 Mbps). This is the recommended solution.

Based on the un-compacted layouts made so far it is not possible to integrate the circuit on one chip. Of the circuits designed, the CSU will require the highest data pin number, 160. The pin requirements for this alternative are discussed in more detail in appendix K.

21.3 Memory requirements.

The conclusion of the analysis of buffer requirements is that two design goals were pushed: the highest bit number possible per area, and the highest clock rate possible. Seven different buffer structures, some standard and some original, have been investigated and designed by the author and his students. Three of these have been processed in several versions. For high density of data, regular RAM structures were found to be the best solution. To hide low clock rate the

parallel width was increased. The buffer found to give the highest clock rate was a fall-through FIFO based on single-phase clock elements. The layout also had a high density compared to the other structures. The layout is given in appendix G. One of the other solutions, an asynchronous FIFO, has been presented in chapter 18. The other buffer structures are not presented in this thesis.

In general, as much as possible of the available chip space should be used for memory. Since single port RAM gives the smallest area per bit, this buffer structure should be chosen. The flow control system and the architecture chosen for SWIPP require a minimum buffer of $N_{ch}2Ct_{fcrt}$ as defined in section 5.5 of this thesis. If the available buffer space is below the minimum requirement, either the number of channels has to be reduced, the flow control strategy changed, the bandwidth reduced or the physical distance between net nodes reduced. With the architecture and bandwidth defined for SWIPP, a maximum physical distance of 250m requires a total buffer of at least 8kbyte for the entire switch. A physical distance of 50 % of the maximum requires 50 % of the buffer space. Buffer space above the minimum value makes the queueing more local.

We would like to keep the average queued data amount local. The average queue length depends on the average and variation of message lengths[3], the average and variation of rate at which messages arrive, and the bandwidth. This is difficult to predict since both message lengths and arrival rates may change with application. The network may try to control this through network protocols, topology and routing algorithms.

Thus, buffer usage is difficult to predict. For the SWIPP architecture, the buffer size should generally be as large as possible.

21.4 Quantum price for integrated circuits.

The switch architecture described in this thesis may, with some change of lay-out, be integrated on one circuit. The solution requiring least work and making this possible, is an implementation in a $0.8\mu m$ BiCMOS technology.

Table 21.2 shows some quantity prices of integrated circuits. In a quantity of 10 000 $100mm^2$ BiCMOS circuits in a $0.8\mu m$ technology would have a price of 60 USD each. With a maximum integrated solution for optical fibre, the dominating additional cost would be the cost of the light sources. Optical laser sources for the bit rates chosen are very expensive. For lasers with a bit rate of 622 Mbps the price would be around 200 USD for each. A laser from HP with a bit rate of 2.5 Gbps would have an expected price of 1200 USD presuming quantities of 10 000.

21.5 The SWIPP project: Verification, contributions and novelty.

In the following, an overview is given of this thesis and the SWIPP switch, with focus on the author's contribution, what is believed to be novel[4] and verification strategy.

[3] Or group of packets transmitted together.

[4] In every scientific work the author(s) should state what he/they believe is novel (unique). What really is novel is a difficult question. It is easier to state what is original (not copied). What is regarded as novel depends on how eager and thorough the authors have been in searching for papers on similar work. It also depends on their ability to appreciate their own work. Thus, what some think is ordinary development, others think is an invention. In other cases the same people may have opposite opinions.

	Quantity	$10mm^2$	$50mm^2$	$100mm^2$
$1.2\mu m$ CMOS	10 000	1.5	10	25
	50 000	1.25	8	22
	100 000	1	7.5	20
$1.2\mu m$ BiCMOS	10 000	2.5	18.5	55
	50 000	2	15	45
	100 000	2	15	42
$0.8\mu m$ CMOS	10 000	2	15	45
	50 000	1.5	12.5	35
	100 000	1.5	12	35
$0.8\mu m$ BiCMOS	10 000	2.5	20	60
	50 000	2	17	50
	100 000	2	16	45

Table 21.2: Typical quantity prices on integrated circuits in USD.

The SWIPP project has been taking place at the Microelectronics Group at the Department of Informatics, University of Oslo, since 1987. The target of the project has become implementation of a switch and a protocol engine. The SWIPP project is under the supervision of Professor Oddvar Søråsen and Professor Yngvar Lundh. Professor Oddvar Søråsen and Professor Yngvar Lundh have mainly focused on the Protocol Engine.

The author of this thesis has guided the analysis and development of the SWIPP switch. The main work has been done by the author and four master students under his supervision. At an earlier stage, a number of undergraduate students also contributed, with short-time projects. A post doctor research scientist contributed on the serial/parallel converters at the switch I/O.

The project has received foundation from the Norwegian Research Council for some scholarships for PhD students and a post doctor student, and for processing of circuits. Except for this, the project has no external connections or foundations.

21.5.1 Verification and verification strategy.

SWIPP has evolved with several sub-studies, investigating different aspects of switch and network design. This has been done to achieve a better basis for understanding. Thus the work has not followed one streamlined description for one goal. Neither have the subdesigns been verified according to a common streamlined verification strategy.

Some global descriptions of the entire switch have been developed. The VHDL model is partially discussed in this thesis. The VHDL code was made to be a high level description for verification of flow control, address pattern and Input and Output Ports. It should also be a description making it possible to simulate several switches together in a reasonable time. The VHDL model has been implemented close to schematics and layouts when necessary for the purpose. It has not been extracted from schematics/layouts, and no automatic comparison has been performed. Thus the VHDL code has not been verified as a correct interpretation of the designed switch layout. A complete listing of the VHDL code would require approximately 100 pages. It is not included in this thesis but is available on Web under the SWIPP project at the Department of

Informatics.

Queuing considerations and switch performance analysis have been verified through C and Simula programs or through mathematical analysis. For verification, simulation results have been compared when possible. The main part of the simulations presented, are given in appendix I and in chapter *13 The arbitration logic of the CSU.*

Almost all modules (i.e. CSUs, Input Ports, Output Port, Elastic FIFOs and Optical Modules) have been verified individually, with electrical (analogue) simulations based on parameters extracted from layout. One exception is the 16×16 CSU described in appendix D. Parts were thoroughly simulated with SPICE and other electrical simulators. Especially the entrance block and the arbiter were carefully simulated and redesigned several times to allow higher clock rate. The total CSU was simulated with a logical simulator. A layout was generated with an auto-place-and-route tool from a $1.2\mu m$ standard cell library designed by the author[5]. The other exception is the "Write pointer transfer logic" in 18.2 where a two-port RAM is proposed used as elastic buffer. The schematic as described in 18.2 is not simulated with an electrical simulator. The structure is described in VHDL and correct operation at a logical level is verified. The metastability problem is thoroughly analysed by one of the author's master students in chapter seven of [86].

The 4×4 CMOS CSU and the 4×4 ECL CSU were processed and later measured. Measurements were done mainly by the students themselves. The measurements indicated almost correct logical operation, but a lot of instability due to the high clock rate.

21.5.2 The author's contribution, and what is believed to be novel.

The main structure of the input buffered switch architecture is well known and mentioned in most network books. Several proposals to how the architecture and architecture parts can be implemented, have been published by others. Some of these papers has been mentioned in connections with description of the sub-modules throughout this thesis. In the following some of the design solutions not found other places will be emphasised.

21.5.3 The CSU (Central Switch Unit)

An early version of schematics and some layouts were developed by the author. The implementation was found to be too slow and new designs were initiated. These two circuits, one in CMOS and one in ECL, were designed by two master students under the author's supervision. The ECL design was partially based on a preliminary study by the author ([77] [30]). The circuits were processed and later tested. These circuits had 4 input and 4 output channels. A new 16 input and 16 output channel CSU (appendix D) was designed by the author alone. This design consists of schematics and an auto-place-and-route layout as described under the discussion of verification above. The simulations were also done by the author alone.

Some schematics of detailed switch implementations have been found in literature. (The references are given in connection with the discussions of the specific modules) Our study for clever schematics of a switch architecture may have resulted in novel solutions valuable for others. An example is the fast arbiter in D.6 and D.8.

[5]Some of the wires should be re-routed to support higher clock rate. This concerns especially the clock signal and some of the wires between the most distant gates.

21.5.4 Circuitry for serial/parallel conversion at the switch I/O s.

An early investigation of serial to parallel converters where done by the author together with a post doctor research scientist. This resulted in a poster and some reports ([66] [38]), but no design. Under the author's supervision a master student made a design consisting of schematics and layout ([94]).

21.5.5 Input Port buffer.

An important element in the switch architecture with a significant influence on performance and chip area is the main buffers positioned in the Input Ports. At the beginning of this project 6-7 different buffer architectures where proposed by the author and implemented in projects for undergraduate students. The main design goals for these buffers were to be elastic andcompact, and have low latency. Low latency meant short time to propagate data from inside the buffer to output of buffer, or directly from input to output when the buffer is empty. All of these architectures required more space and had higher latency than implementations developed later. These architectures are not presented in this thesis.

Later a master student has, under the author's supervision, implemented an asynchronous FIFO ([86]). This solution is briefly described in chapter *18 Implementation of the elastic FIFO* and based on a paper by Ivan. E. Sutherland ([105] [104])

Since high clock rate and dense buffer is difficult to combine, the author's conclusion is that two types of buffers should be implemented:
- A high density buffer for large buffer requirements, and
- a high clock rate FIFO for small buffer requirements.

In the first buffer type, the design goal is only high density. Low internal clock rate is hidden behind high speed expanders /concentrators on the buffer input / output. The bandwidth can be scaled to fulfil any requirement. The main disadvantage is high latency. Since standard RAM compilers offer the smallest area per bit, the buffer core is based on such RAM structures. The structure is discussed in chapter *18 Implementation of the elastic FIFO*. The architecture is an original work by the author, not based on literature, except for the general understanding of meta-stability in the master thesis by one of the author's students ([86] chapter 7).

The high speed FIFO (layout in appendix G.1) was designed by the author of this thesis alone. It is a fall-through FIFO operating on a true single-phase clock. The memory cell is based on [112]. Modification of the memory cell and the development of the fast control circuitry is the author's work. The author has not found any other FIFO architectures with a smaller fall-through latency and a higher clock rate. This statement is primarily based on standard VLSI text books like [109] and the last volumes of IEEE Journal of Solid State Electronics. Extra effort has been put in making the layout as dense as possible. Thus it has become, except for the RAM core, the most dense part of the switch layouts so far.

21.5.6 Input and Output Port.

The design and implementation of the Input and Output Port has been done by the author alone. Several proposals were developed at a schematic level. They all required too long clock periods. Except for the state table in appendix F, this work is not presented in this thesis. New

proposals based on pipelining was developed. Through several redesigns their logical function were improved with a digital schematic and simulation tool[6]. After correct logical function was found, the ASIC layout was implemented with Magic[7] Smaller sections were simulated with the electrical simulator SPICE[8]. The simulations where performed on netlists extracted from the layout. Transistor structures where changed and transistor sizes on critical paths where modified to improve speed. Capacitive load on heavily loaded lines were extracted and buffers strengthened. Thus all signal paths from clock edge to clock edge were simulated. Since the electrical simulator at the present time could not simulate entire ports, the top level simulations were done with a logical simulator.

Since no one has designed an Input Port and an Output Port with exactly the same function, the design is obviously both original and unique. If some general knowledge should be drawn from the design, this probably has to be as an example of how pipelining can be implemented on a data stream.

21.5.7 Overview of switch performance, traffic simulations and queuing theory.

Some of the more heavy work by the author has been put in traffic simulations of simple switch architectures. The work was done by the author alone with no contributions from others (except for the help from one of L. Kleinrock's basis books in queuing theory [61]). Models of input buffered switches, with bypass, with multiple plans and with different sizes have been developed. These models have been simulated with different traffic patterns. Some of the heavy simulation work is presented in appendix I. Appendix E contains some topology characteristics based on a M/M/1 traffic model. The author also made some simulation code to find the behaviour of unfair arbiters for different load levels.

The simulations and calculations have significantly increased this author's understanding of the relations behind switch performance and behind topology performance. During and after the work was finished this author learnt to know several papers presenting similar topics. While the author's work covers both M/M/1 and M/D/1 traffic patterns with a weight on M/M/1, many of the papers found focus on M/D/1 alone. Thus the work of this author is original, but it may be too close to other published work to be considered unique.

When switch architectures are compared in the literature, they are often characterised with random uniform destination distributions. This is far from the true behaviour of real networks. The author also did some simulations on non-uniform destination distributions. The simulations were performed with common concentrated traffic on a few destinations, and with different and unique destination distributions for each input channel. The result of these simulations are not presented in this thesis, but has contributed to the author's understanding of switches and topologies.

21.5.8 Network protocols.

The author has done some literature studies to achieve an understanding of network protocols and their influence on network performance. Some of this work is presented in the first chapters

[6] LOG from California Institute of Technology.
[7] From the University of Berkeley, California, USA.
[8] Also from the University of Berkeley, California, USA.

of this thesis.

21.5.9 SWIPP publications.

The SWIPP project has resulted in several publications. A selection is given in the following:

Optomodules: [94], [66], [79] and [75].
Switch: [7], [54], [62], [70] and [63].
Protocol Engine: [9] and [28].
Asynchronous FIFO: [86].
Top level description: [67], [78] and [37].
CERN Applications: [71], [96]and [76].
SWIPP bridges: [87].
Multicomputer: [36].
Technology studies: [30] and [77].

Of these, the author of this thesis has been author or co-author on the following work: [97], [66], [78], [77], [30], [96], [76] and [37]
The following master theses have been written under the supervision of the author: [94], [87], [70], [63] and [62]

21.6 Further work.

The first goal should be to design the very few modules missing and compact the layout of the ones requiring too much space. This should give a complete switch on one circuit in the near future.

The second goal should be to study the architecture to see how bandwidth can be kept or increased with less power consumption.

It is also important to make the layout more scaleable so that reduced space by scaling to new small geometric technologies can be achieved. This should be the third goal.

A fourth goal could be to study switch architectures with buffer reservation strategies. A solution between SWIPP and Telegraphos could offer better performance without their disadvantages.

Chapter 22

Conclusion.

The aim of the concept of which this thesis is a part, is to add increased performance to a system of homogenous or heterogeneous computing resources. The increased performance is achieved through a high performance network with a global and distributed operating system. The function of the operating system is to utilise the collected computing power better.

This thesis has focused on the network part. In such a system, several design criteria for a network are possible. During this work we have found some promising design criteria, which are listed in the beginning of the previous chapter. The thesis describes the architecture of a switch designed using these criteria. The switches are small, can be placed almost everywhere, and have a relatively high performance. They are believed to be usable for a number of applications. They are clearly competitive with many commercially available switches, often having a little larger performance but a much larger physical size.

The author believes that there is a large potential both commercially and academically for networks like this. This is confirmed by the number of industrial companies and the many universities with research projects in this field. The SWIPP concept is different and should absolutely have a potential and a position among the others.

Bibliography

[1] D. S. Ahn and M. J. Lee. *Cell Loss Analysis and Design Trade-Offs of Nonblocking ATM Switches with Nonuniform Traffic*, volume Vol.3 of *No.2*. IEEE/ACM Transactions on networking, April 1995. pp 199-210.

[2] J. Alvarez, H. Sanchez, G. Gerosa, and R. Countryman. *A Wide-Bandwidth Low-Voltage PLL for PowerPC^{TM} Microprocessors*, volume Vol. 30 of *No. 4*. IEEE Journal Of Solid-State Circuits, April 1995. pp. 383-391.

[3] T.E. Anderson, D.E. Culler, and D.A. Patterson. *A Case for NOW (Networks of Workstations)*. IEEE Micro. February 1995, http://now.cs.berkeley.edu/.

[4] L.I. Andersson, B.G.R. Rudberg, P.T. Lewin, M.D. Reed, S.M. Planer, and S.L. Sundaram. *Silicon Bipolar Chipset for SONET/SDH 10Gb/s Fiber-Optic Communication Links*, volume Vol. 30 of *No. 3*. IEEE Journal Of Solid-State Circuits, March 1995. pp. 210-218.

[5] E. A. Arnould, F. J. Bitz, E. C. Cooper, H.T.Kung, R. D. Sansom, and P. A. Steenkiste. *The Design of Nectar: A Network Backplane for Heterogenous Multicomputers*. The Proceedings of the Third International Conference on Architectural Support for programming Languages and Operating Systems, April 1989. ACM.

[6] H.B. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI*. Addison-Wesley Publishing Company, 1990. ISBN 0-201-06008-6.

[7] P.M. Lie Baltzersen. *Svitjsenoder for et Pakkesvitsjet Multiprosessornett*. Master thesis (in Norwegian), Department of Informatics, University of Oslo, August 1989.

[8] G. K. Bhattacharryya and R. A. Johnson. *Statistical Concepts and Methods*. John Wiley & Sons, 1977.

[9] I. Blekastad and M. Hagen. *Protokollmaskin, en kommunikasjonsenhet for en høyhastighets multiprocessor* . Master thesis (in Norwegian), Department of Informatics, University of Oslo, Jan 1989.

[10] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. *Myrinet: A Gigabit-per-second Local Area Network*. IEEE Micro, February 1995. pp. 29-36, URL: http://www.myri.com/.

[11] S. Borkar e.a. *Supporting Systolic and Memory Communication in IWarp* . Proc. of the 17th Int. Symp. on Computer Architecture, ACM SIGARCH vol 18, no.2, June 1990, pp. 70-81.

[12] Bull/SGS-Thompson. *STRINGS/BULLIT*. January 1996, pp 32-44.

[13] W.E. Burr. The FDDI optical data link. *IEEE Communication Magazine*, Vol. 24:pp. 18–23, May 1986.

[14] G. Chesson and L. Green. *Inmos H1 transputer, Product preview.* September 1990.

[15] I. Cidon and I. Gopal. *Paris: An approach to integrated high-speed private networks.* Intl. Journal Digital and Analog Cabled Systems, Vol. 1, 1988. pp. 77-85.

[16] I. Cidon, I. Gopal, J. Janniello, and M. Kaplan. *The plaNET/ORBIT High Speed Network.* RC-18270, IBM, 1992.

[17] D. Cohen, G. Finn, R. Felderman, and A. DeSchon. *ATOMIC: A Low-Cost, Very-High-Speed LAN.* USC/Information Sciences Institute, Undated, received 1995.

[18] E. C. Cooper, P. A. Steenkiste, R. D. Sansom, and B. D. Zill. *Protocol Implementation on the Nectar Communication Processor.* School of Computer Science, Carnegie Mellon University, Pittburgh, Pennsylvania 15213, USA, 1990.

[19] M. Cooperman. *CMOS Gigabit-per-Second Switching*, volume Vol. 28. of *No.6.* IEEE Journal of Solid-State Circuits, June 1993. pp 631-639.

[20] M.D. Dahlin. *Cooperative Caching: Using Remote Client Memory to Improve File System Performance.* Proceedings of the First Symposium on Operating System Design and Implementation. 1994, http://www.cs.berkeley.edu/.

[21] W. Dally and C. Seitz. *Deadlock-Free Message Routing in Multiprocessor Interconnection Networks.* IEEE Trans. on Computers, vol. C-36, no.5, May 1987, pp.547-553.

[22] W. J. Dally. *Virtual-Channel Flow Control* . Proc. of the 17th Int. Symp. on Computer Architecture, ACM SIGARCH vol. 18, no.2, May 1990, pp. 60-68.

[23] R. G. Daniels. *A participant's Perspective: The "father" of the 6805 MCU relates his rule in the history of the microprocessor.* IEEE Micro, January 1996, pp 21-31.

[24] E.E. Davidson. *Electrical design of a high speed computer packaging system.* IBM Journal of Research and Development, May 1982. vol.26, no.3, pp.349-361.

[25] J. Demmel and S. Smith. *Parallelizing a Global Atmospheric Chemical Tracer Model.* Symp. High Performance Computing and Communications, 1994.

[26] Digital Equipment Co. *GIGAswitch/ATM Networking Switch.* Nov. 1994, URL: file: //ftp.digital.com/pub/Digital/info/infosheet/EC-F3924-42.ps.

[27] J. Dunning, G. Garcia, J. Lundberg, and E. Nuckolls. *An All-Digital Phase-Loacked Loop with 50-Cycle Lock Time Suitable for High-Performance Microprocessors*, volume Vol. 30 of *No. 4.* IEEE Journal Of Solid-State Circuits, April 1995. pp. 412-422.

[28] R. Esvall and A.S. Bonde. *Protokollmaskinen PE. En nettuavhengig kommunikasjonsenhet tilpasset SWIPP-konseptet* . Master thesis (in Norwegian), Department of Informatics, University of Oslo, Aug 1992.

[29] E.C. Foudriat, K. Maly, C.M. Overstreet, S. Khanna, and F. Paterra. *A carrier sensed multiple access protocol for high data rate ring networks.* ACM Computer Communication Review, Vol. 21, No. 2, April 1991. pp. 59-70.

[30] A. Gakkestad, L. Hanssen, A. Kjensmo, H. von der Lippe, and J.M. Østby. *Analog/Digital BiCMOS.* Research Note (in Norwegian), Senter for industriforskning, August 1991.

[31] L. Geppert. *Technology 1996 — Solid State.* IEEE Spectrum, January 1996. pp 51-55 (p53).

[32] J. N. Giacopelli, J.J. Hickey, W.S. Marcus, W.D. Sincoskie, and M. Littewood. *Sunshine: A High-Performance Self-Routing Broadband Packet Switch Architecture* . IEEE Journal on Selected Areas in Communications, Volume 9, No. 8, October 1991, pp. 1289-1298.

[33] R. B. Gillett. *Memory Channel Network for PCI.* IEEE Micro, February 1996. pp 12-18.

[34] L. R. Goke and G. J. Lipovski. *Banyan networks for partitioning multicomputer systems.* Proc. First Annual Computer Architecture Conf., pp. 21-28, Dec 1973.

[35] L. Goldberg. *NIC-On-A-Chip Promises To Push ATM To The Desktop.* Electronic Design, Number 6 1995. pp. 175-176.

[36] Ø Gran-Larsen. *Development and emulation of interaction mechanisms for a heterogeneous multicomputer.* Ph.D. thesis, Department of Informatics, University of Oslo, Oct 1991.

[37] Ø. Gran-Larsen, L. Guoning, Y. Lundh, O. Søraasen, and J. M. Østby. *SWIPP - Switched Interconnection of Parallel Processors.* Department of informatics, University of Oslo, P. O. Box 1080 Blindern, N-0316 Oslo, NORWAY
Undated, available from the author.

[38] L. Guoning, J.M. Østby, O. Søråsen, and Y. Lundh. *An Optical Data Link for Multicomputer Systems.* Research Report, Department of Informatics, University of Oslo, ISBN 82-7368-059-2, 1991.

[39] M. Henrion, K. Schrodi, D. Boettle, M. De Somer, and M. Dieudonne. *Switching network architecture for ATM based broadband communications.* ISS'90, Stockholm, June 1990.

[40] L.J. Herbst. *High Speed Digital Electronics.* Prentice Hall, 1989. ISBN 0-13-388943-2.

[41] M. Hirata, S. Kikuchi, M. Suzuki, and N. Yamanaka. *A Circuit Design for 2-Gbit/s Si Biplar Crosspoint Switch LSI's*, volume Vol. 25 of *No. 1.* IEEE Journal of Solid-State Circuits, February 1990. pp. 155-159.

[42] M. G. Hluchyj and M. J. Karol. *Queueing in High-Performance Packet Switching.* IEEE Journal On Selected Areas In Communications, Vol. 6, No. 9, December 1988. pp. 1587-1597.

[43] A. Huang. *The relationship between STARLITE, A wideband digital switch and optics.* in Proc. ICC'86, Toronto, Canada, pp. 1725-1729, June 1986.

[44] A. Huang and S. Knauer. *Starlite: A wideband Digital Switch.* GLOBECOM'84, pp. 121-125, Nov. 1984.

[45] J. Y. Hui. *Switching and traffic theory for integrated broadband networks.* Kluwer Academic Publishers, 1990. ISBN 0-7923-9061-X.

[46] C. Hunt. *TCP/IP Network Administration.* O'Reilly & Associates, Inc., 1992. ISBN 0-937175-82-X.

[47] IEEE. *Distributed Queue Dual Bus (DQDB) Subnetwork of a Metropolitian Area Network (MAN).* Institute for Electrical and Electronics Engineers, Piscataway, NJ, 3 July 1991.

[48] IEEE (The Institute of Electrical and Electronics Engineers). *Std. 1596-1992, Scalable Coherent Interface (SCI).* Piscataway, N.J., 1992.

[49] INMOS Ltd. *IMS C104 Packet Routing Switch - Preliminary Datasheet.* 1000 Aztec West, Bristol, BS12 4SQ, UK, June, 1993.

[50] Internet - Network Working group. *RFC (Request For Comments) 1752, The recommendation for the IP next generation protocol (IPv6/IPng).* S. Bradner, Harvard University, A. Mankin, ISI, January 1995. ftp://ds.internic.net/rfc/rfc1752.txt.

[51] Internet - Network Working Group - Audio-Video Transport Working Group. *RFC (Request For Comments) 1889, RTP: A transport protocol for real-time applications.* H. Schulzrinne, GMD Fokus, S. Casner, Precept Software, Inc., R. Frederick, Xerox Palo Alto Research Center, V. Jacobson, Lawrence Berkeley National Laboratory, January 1996. http://info.internet.isi.edu:80/in-notes/rfc/rfc1889.txt.

[52] Sunil P. Joshi. *High-Performance Networks: A Focus on the Fiber Distributed Data Interface (FDDI) Standard.* IEEE Micro, June 1986. pp. 8-14.

[53] E. Juliussen. *Small computers.* IEEE Spectrum, January 1995. pp 44-47.

[54] F.R. Karlsen. *Et høyhastighets, fiberoptisk flerprosessornett.* Master thesis (in Norwegian), Department of Informatics, University of Oslo, Oct 1989.

[55] M. Katevenis. *Telegraphos: High-Speed Communication Architecture for Parallel and Distributed Computer Systems.* Institute of Computer Science (ICS), Science and Technology Park, Heraklio, Crete, POBox 1385, GR-711-10, Greece, 1994.

[56] M. Katevenis, P.Vatsolaki, A. Efthymiou, and M. Stratakis. *VC-level Flow Control and shared Buffering in the Telegraphos Switch* . Proceedings of the Hot Interconnects III Symposium, Stanford Univ., CA, USA, Aug. 1995, URL: file://ftp.ics.forth.gr/tech-reports/1995/1995.HITI.VCflowCtrlTeleSwitch.ps.gz.

[57] M. Katevenis, D. Serpanos, and E. Spyridakis. *Credit-Flow-Controlled ATM versus Wormhole Routing* . Technical Report FORTH-ICS/TR-171, ICS, FORTH, Heraklio, Crete, Greece, July 1996, URL: file://ftp.ics.forth.gr/tech-reports/1996/1996.TR171.ATM_ vs_ Wormhole.ps.gz.

[58] M. Katevenis, D. Serpanos, and P. Vatsolaki. *ATLAS I: A General-Purpose, Single-Chip ATM Switch with Credit-Based Flow Control* . Proceedings of the Hot Interconnects IV Symposium, Stanford Univ., CA , USA, Aug. 1996, pp. 63-73,, URL: file://ftp.ics.forth.gr/tech-reports/1996/1996/.HOTI.ATLAS_ I_ ATMswitchChip.ps.gz.

[59] M. Katevenis, P. Vatsolaki, and A. Efthymiou. *Pipelined Memory Organization for High Performance Switching and Buffering.* Institute of Computer Science (ICS), Science and Technology Park, Heraklio, Crete, POBox 1385, GR-711-10, Greece, 1974.

[60] M. Katevenis, P. Vatsolaki, and E. Efthymiou. *Pipelined Memory Shared Buffer for VLSI Switches.* Proceedings of the ACM SIGCOM '95 Conference, Cambridge, MA USA, 30/8 - 1/9. 1995, pp. 39-48, URL: file://ftp.ics.forth.gr/tech-reports/1995/1995.SIGCOMM95.PipeMemoryShBuf.ps.gz.

[61] L. Kleinrock. *Queueing Systems, Vol 2: Computer Applications.* John Wiley: New York, 1976. ISBN 0-471-49111-X.

[62] J. E. Kvarme. *Testrapport: Testing av 4x4 crossbar-svitsj (CMOS).* (Supervised by this author), Avdeling for digitalteknikk/mikroelektronikk, Institutt for Informatikk, Universitetet i Oslo,
In norwegian. Availeble from the author of this thesis., May 1994.

[63] J.E. Kvarme. *Konstruksjon av crossbar-svitsj i høyhastighets CMOS for multiprosessor nettverk.* Master thesis (in Norwegian), (Supervised by this author), Department of Informatics, University of Oslo, Nov 1993.

[64] R. Lamaire and D. Serpanos. *Two-Dimensional Round-Robin Schedulers for Packet Switches with Multiple Input Queues.* IEEE / ACM Trans. on Networking, vol.2, no. 5, October 1994, pp. 471-482.

[65] H.-S Lee, C. G. Sodini, and K. M. Ware. *A 200-MHz CMOS Phase-Locked Loop with Dual Phase Detectors*, volume Vol. 24 of *No. 6.* IEEE Journal of Solid-State Circuits, December 1989. pp. 1560-1568.

[66] G. Liao, J.M. Østby, O. Søråsen, and Y. Lundh. *Self Synchronizing Data Transfer Scheme for Multicomputers* . Poster for **The 1991 International Conference on Parallel Processing**, St. Charles, Illinois, Aug 1991.
Also available as Preprint 1991-3, May 27, 1991. Department of Informatics, University of Oslo.

[67] Y. Lundh. *Skisse av multiprosessorstruktur* . Internal note (in Norwegian), Department of Informatics, University of Oslo, Oct 1987.

[68] Y. Lundh and O. Søråsen. *SWIPP - A Multicomputer Framework for Bulk Synchronous Parallel Computing.* 1997.

[69] R. P. Martin. *HPAM: An Active Message layer for a Network of HP Workstations* . University of California at Berkeley.

[70] M. Moe. *Konstruksjon av crossbar-svitsj i ECL for multiprosessor nettverk.* Master thesis (in Norwegian), (Supervised by this author), Department of Informatics, University of Oslo, Aug 1993.

[71] F.B. Nilsen. *Application of a switched interconnection concept for instrumentation at a high-energy physics experiment* . Master thesis, Department of Informatics, University of Oslo, May 1993.

[72] N.N. . Private communication, explained to me at a conferance in 1991.

[73] M. Noakes, D. Wallach, and W. Dally. *The J-Machine Multicomputer: An Architectural Evaluation* . Proc. of the 20th Int. Symp. on Computer Architecture, ACM SIGARCH vol. 21, no.2. May 1993, pp. 224-235.

[74] J.B. Nysæther and E.U. Poppe. *Thermal properties of flip chip bonded silicon hybrids.* SINTEF Instrumentation, 1995.

[75] H. Olsen. *Optisk prosessorkommunikasjon.* Master thesis (in Norwegian), Department of Informatics, University of Oslo., May 1992.

[76] J. M. Østby and O. Søråsen. *A Packet-Switched Network for Data Readout from the LHC Inner Detector* . Computing In High Energy Physics '94, San Francisco, USA, Oral presentation, pp 173-176, 21-27 April 1994.

[77] J.M. Østby. *Litt om BiCMOS og GaAs i digitale anvendelser.* Lecture notes,, April 91.

[78] J.M. Østby. *Definition of Some Cuts in the Multicomputer Network: Draft no 2.0.* Internal note, Department of Informatics, University of Oslo,, Nov 1990.

[79] J. Østenstad. *Modul for fiber-optisk kommunikasjon i et høyhastighets multiprosessor-nettverk* . Master thesis (in Norwegian), Department of Informatics, University of Oslo, May 1992.

[80] Hewlett Packard. *Low Cost Gigabit Rate Transmit/Receive Chip Set.* Technical Data, Received 1996.

[81] C. Partridge. *Gigabit Networking.* Addison-Wesley Professional Computing Series, 1993. ISBN 0-201-56333-9.

[82] M. De Prycker. *Asynchronous Transfer Mode.* Ellis Horwood, 1993. ISBN 0-13-178542-7.

[83] M De Prycker, M. De Somer, M. Watteyne, J. Vandedrinck, J. Van Vyve, and M. Van Laethem. *An ATM Switching Architecture with Intrinsic Multicast Capabilities for the Belgian Broadband Experiment.* ISS'90, Stockholm, May 1990.

[84] B. Razavi. *A 2.5Gb/s 15mW Clock Recovery Circuit*, volume Vol. 31 of *No. 4.* IEEE Journal Of Solid-State Circuits, April 1996. pp. 472-480.

[85] B. Razavi, K. F. Lee, and R. H. Yan. *Design of High-Speed, Low-Power Frequency Dividers and Phase-Locked Loops in Deep Submicron CMOS*, volume Vol. 30 of *No. 2.* IEEE Journal Of Solid-State Circuits, February 1995. pp. 101-109.

[86] P.T. Røine. *Asynchronous FIFO Buffer for Multicomputer Applications.* Master thesis, (Supervised by this author), Department of Informatics, University of Oslo, May 1994.

[87] Ø Roseth. *Brokobling mellom et multicomputernettverk og et lokalt nettverk* . Master thesis (in Norwegian), (Supervised by this author), Department of Informatics, University of Oslo, Aug 1991.

[88] D. Scales, M. Burrows, and C. Thekkath. *Experiences with Parallel Computing on the AN2 Network*, volume IEEE Computer Socity Press, Los Alamitos, Calif. to be published in *Proc. 10th IEEE Int'l Parallel Processing Symp.*, 1996.

[89] M. D. Schroeder, A. D. Birrell, M. Burrows, H. Murray, R. M. Needham, T. L. Rodeheffer, E. H. Satterthwaite, and C. P. Thacker. *Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links*, volume Digital Equipment Corporation. SRC Research Report 59, 1990.

[90] M.D. Schroeder. *Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-Point Links.* SRC Research Report 59, April 2, 1995. http://www.myri.com/.

[91] C. Seitz et al. *Wormhole Chip Project Report.* Winter, 1985.

[92] H. J. Shin and D. A. Hodges. *A 250-Mbit/s CMOS Crosspoint Switch*, volume Vol. 24 of *No 2.* IEEE Journal Of Solid-State Circuits, April 1989. pp. 478-486.

[93] R. Shuford. *An introduction to fiber optics.* Byte, Desember 1984.

[94] M. R. Sivasothy. *Electrical to Optical Module and Optical to Electrical Module for optic communication.* Master thesis, (Supervised by this author), Department of Informatics, University of Oslo, May 1993.

[95] M. Slater. *The Microprocessor Today.* IEEE Micro, January 1996, pp 32-44.

[96] O. Søråsen, Y. Lundh, F. B. Nilsen, E. Nygård, and J. M. Østby. *A High Performance Data Driven Packet-Switching Network for Detector Data Readout and Event Building in an LHC Inner Detector Experiment* . IEEE Eight Conference on Real-Time Computer Applications in Nuclear, Particle and Plasma Physics, June 8-11, 1993
Also available from: IEEE Transactions on nuclear science, Vol. 41, No. 1, February 1994. pp 246-251.

[97] O. Søråsen, F.B. Nilsen, E. Nygård, and J.M. Østby. *A Switched Interconnection Network for Large Scale Instrumentations.* **ATLAS Internal DAQ Note 06,** October 1992,, October 1992. **RD-20 Note 10,**.

[98] R. Souza, P. Krishnakumar, C. Ozveren, R. Simcoe, B. Spinney, R. Thomas, and R. Walsh. *GIGAswitch System: A High-Performance Packet-Switching Platform.* Digital Technical Journal, DEC,, vol.6, no.1, Winter 1994, pp.9-22.

[99] W. Stallings. *Data and Computer Communications.* Collier Macmillan Publishers, 1988. ISBN 0-02-415451-2.

[100] P. Steenkiste. *Analyzing Communication Latency using the Nectar Communication Processor.* School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania 15213, USA, 1995.

[101] P. Steenkiste. *A Systematic Approach to Host Interface Design for High-Speed Networks.* School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, Pennsylvania 15213, USA, Undated, received 1995.

[102] P. A. Steenkiste, S. J. Schlick B. D. Zill, H.T. Kung, J. Hughes, B. Kowalski, and J. Mullaney. *A Host Interface Architecture for High-Speed Networks.* School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213-3890, USA, 1995.

[103] S.Y. Sun. *An Analog PLL-Based Clock and Data Recovery Circuit with High Input Jitter Tolerance*, volume Vol. 24 of *No. 2.* IEEE Journal Of Solid-State Circuits, April 1989. pp. 325-330.

[104] I. E. Sutherland. *Asynchronous first-in-first-out register structure.* US Patent Pending, 1990.

[105] I. E. Sutherland. *Micropipelines.* Communications of the ACM, 32(6):720-738, June 1989.

[106] S.M. Sze. *Semiconductor devices: Physics and Technology.* John Wiley & Sons, 1985.

[107] A. S. Tanenbaum. *Computer Networks.* Prentice-Hall International Editions, 1988. ISBN 0-13-166836-6.

[108] G. Watson, S. Ooi, D. Skellern, and D. Cunningham. *HANGMAN Gb/s Network.* IEEE Network Magazine, Vol. 6, No. 4, July 1992. pp. 10-18.

[109] N. Weste and K. Eshragian. *Principles of CMOS VLSI Design.* Addison-Wesley Publishing Company, 1988. ISBN 0-201-08222-5.

[110] C.L. Wu and T.Y. Feng. *Tutorial: Interconnection Networks for Parallel and Distributed Processing.* IEEE Computer Socity Press, 1984.

[111] Yu-Shian Yeh, Michael G. Hluchyj, and Anthony S. Acampora. *The Knockout Switch: A Simple, Modular Architecture for High-Performance Packet Switching.* IEEE Journal On Selected Areas In Communications, Vol. 5, No. 8, October 1987. pp. 1274-1283.

[112] J. Yuan and C. Svensson. *High-Speed CMOS Circuit Technique*, volume Vol. 24 of *No. 1*. IEEE Journal Of Solid-State Circuits, February 1989. pp. 62-70.

APPENDIX

Appendix A:
Definitions of symbols and signals

A.1 Symbol definition.

A symbol consists of 9 bits. They are divided into two groups: the data symbols with a leading one and the control symbols with a leading zero.

1	d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0	**Data symbol**
0	c_2	c_1	c_0	f	x	x	x	x	*Control symbol*

The following syntax is used:
d_i is a data bit.
$c_2c_1c_0$ identifies a control symbol with a specific meaning and function.
f is a flow control bit.
a high f signals "start/continue transmission" while
a low f signals "stop transmission".
x is not used or is undefined at this level.

The names of control symbols are written in *italic* while the names of data symbols or data fields only containing data symbols are written with **bold face** letters.

Transmission of symbols on 9-line-wide buses.

Symbols are transmitted in parallell on 9-line-wide buses as given in the format above. Symbols are clocked on the positive clock edge.

Transmission of symbols on 5-line-wide buses.

Symbols are transmitted in two parts on 5-line-wide buses : the first part is transmitted on the high clock level while the second half is transmitted on the following low clock level. The format is as follows:

Clock level	5-line bus					
High	1	d_7	d_6	d_5	d_4	Data
Low	f	d_3	d_2	d_1	d_0	symbol
High	0	c_2	c_1	c_0	x	Control
Low	f	x	x	x	x	symbol

Notice that the flow control bit is transmitted with both data and control symbols.

The bit codes of the control symbols.

		5-line wide bus									9-line wide bus
bop	*beginning of packet*	0	1	0	1	x	f	0	0	0	0 101x 0000
eop	*end of packet*	0	1	0	0	x	f	0	0	0	0 100x 0000
df	*data follows*	0	0	1	0	x	f	0	0	0	0 010x 0000
idle	*idle*	0	0	0	0	x	f	0	0	0	0 000x 0000
A4	*4-bit binar address follows*	0	1	1	0	x	f	a_3^4	a_2^4	a_1^4	a_0^4 not used
A16	*16-bit address mask follows*	0	1	1	1	x	f	a_{15}^{16}	a_{14}^{16}	a_{13}^{16}	a_{12}^{16} not used
	8 midle significant bits	1	a_{11}^{16}	a_{10}^{16}	a_9^{16}	a_8^{16}	f	a_7^{16}	a_6^{16}	a_5^{16}	a_4^{16}
	4 least significant bits	1	a_3^{16}	a_2^{16}	a_1^{16}	a_0^{16}	f	0	0	0	0

The table above shows the bit codes for control symbols as they are transmitted on 5-line-wide and 9-line-wide buses. The *A4* and *A16* control symbols are generated by the Input Port for the CSU. Thus they are never transmitted on a 9-line-wide bus. The *A16* control symbol used for selective broadcast uses 3 clock periods. The symbol part transmitted during the last two clock periods have a leading 1 like the data symbols.

A.2 Packet format.

At the highest level the packet consists of control symbols and data fields.

Data fields are written in **bold face** letters. Their contents are transmitted as data symbols. The control symbols delimiting and enclosing the data fields are written in *italics*.

⇐ Direction of transmission

bop	**switch data field**				*df*	**PE data field**	*eop*
	Jump counter	**forward address list**	**[dummy]**	**backward address list**			

The packet format consists of two data fields and three control symbols. The **switch data field** consists of necessary information used by the switches for packet routing. The **PE data field** consists of the payload carried between the Protocol Engines. The **switch data field** has a maximum length of 8 bytes. The **PE data field** can have any length, but fixed or maximum lengths can be practised by Prototcol Engine software.

The switch data field.

The switch data field consists of four sub-fields: the **jump counter**, the **forward address list**, the optional **dummy field** and the **backward address list**. The **jump counter** and the **dummy field** are 4-bit-nibbles while the **forward address list** and **the backward address list** are sequences of 4-bit-nibbles. The contents of the **jump counter** gives the number of nibbles in the **forward address list**. The total number of nibbles in the **forward address list** and the **the backward address list** is 15 or less. The total number of nibbles in the **switch data field** is made even by adding the **dummy** field depending on the number of nibbles in the three other fields.

Chapter *7: The SWIPP packet and address format* gives a more detailed description of the packet format.

A.3 The names of the flow control signals.

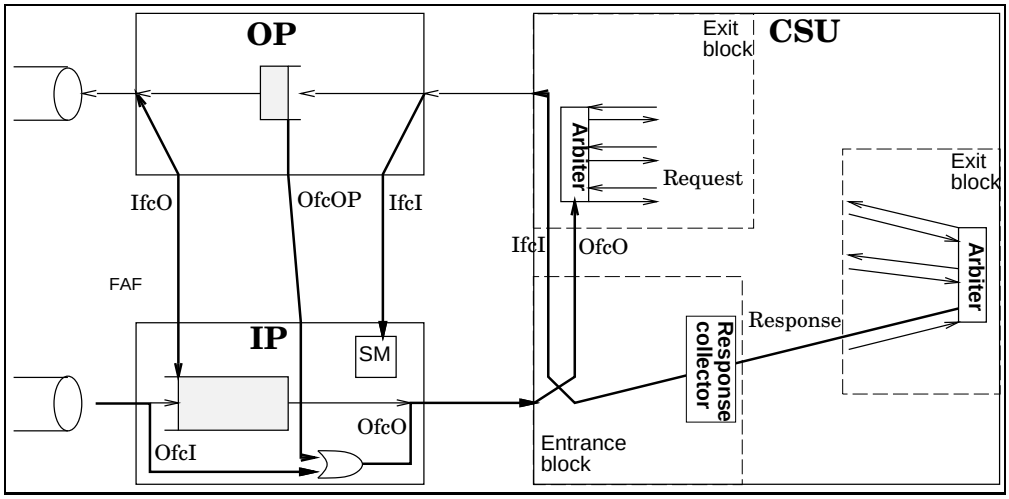


Figure A.1: The flow control signal has different names in different places. The figure illustrates this.

A.3.1 Flow control signals inside an Input and Output Port pair.

In figure A.3 we take a closer look at the flow control signalling inside an IP/OP couple (e.g. the *C* IP and OP in the previous figures).

The **I** packet stream is the packet stream directed towards the CSU. It is passing through the upper part of figure A.3 in the left direction through the Input Port. The flow control signals belonging to this connection is named **IfcI** and **IfcO**.

The **O** packet stream is the packet stream directed from the CSU. It is passing through the Output Port in the right direction in the lower part of the figure. The flow control signals belonging to this connection are **OfcI**, **OfcO** and **OfcOp**.

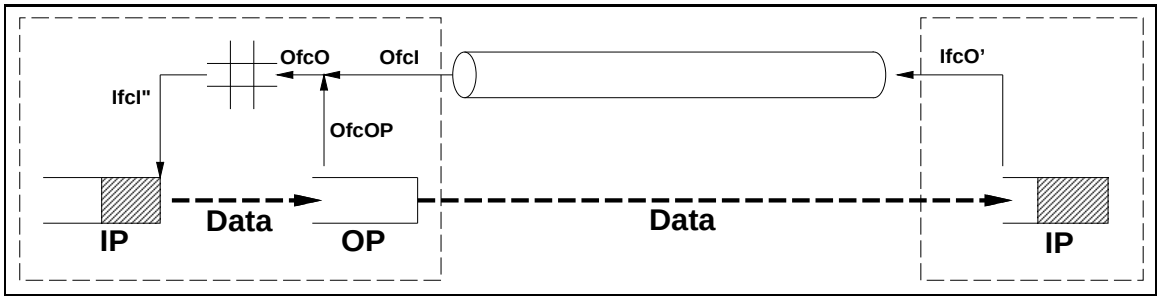


Figure A.2: This figure illustrates the different names of the same flow control signal on the way upstream. The data transmitter is on the left side while the receiver is on the right side. The flow control signal is forwarded in the opposite direction.

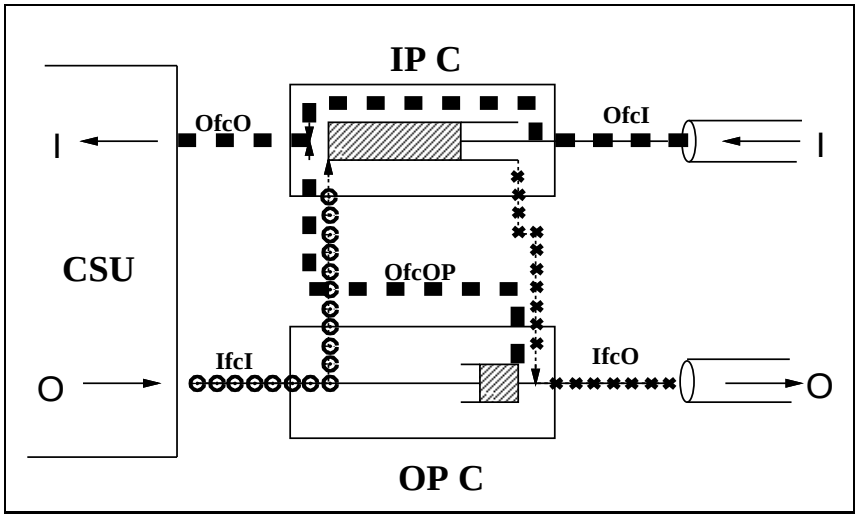


Figure A.3: Flow control signals inside an IP/OP pair. The signals will be described in detail in the following.

The O packet stream directed out from the CSU.

The **O** packet stream is drawn as an arrow in the lower half of figure A.4. It continues in the right direction through the fibre to an Input Port (labelled E) in the next net node. This receiving Input Port (E) generates the flow control signals which are passing through an intermediate Input Port (labelled C). The flow control signal is named **OfcI** as it arrives at Input Port C. The flow control signal bypasses Input Port C, enters the CSU, and arrives at the transmitting Input Port (labelled A) at the other end.

As explained in the previous sections also the Output Port has a small FIFO buffer. If this buffer is filled a little break in the packet stream is needed to empty the buffer. The **OfcOp** flow control signal from the Output Port is used to generate this break. This signal is transmitted to the Input Port where a logical AND is performed between the **OfcOp** signal and the **OfcI** signal. The **OfcO** signal is the result of this logical AND.

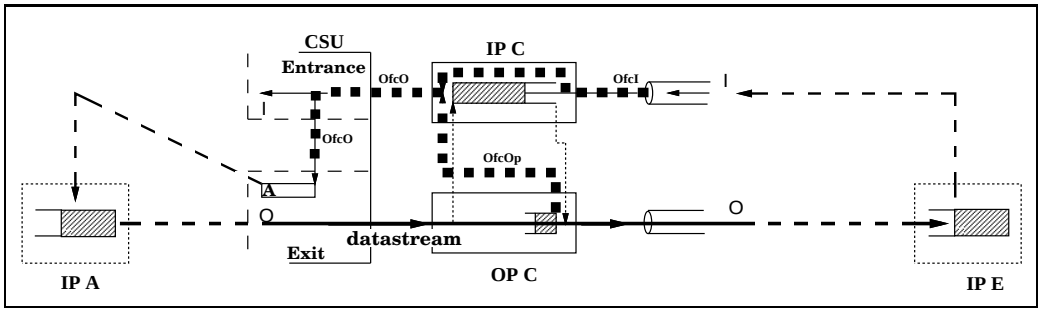


Figure A.4: Flow control signals for packet stream out from the CSU.

The I packet stream directed towards the CSU.

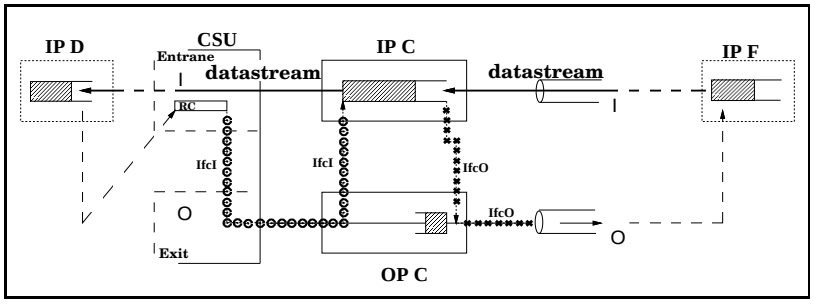


Figure A.5: Flow control signals for packet stream into the CSU.

Incoming data pass through the upper half of figure A.5 in the left direction.

The Input Port in the middle of the figure (labelled C) receives data through the fibre from an Input Port buffer (F) in a remote net node. If the amount of data in the receiving buffer passes the "almost full" mark a flow control signal will be transmitted upstream to the feeding Input Port buffer (F). The flow control signal is named **IfcO** as it leaves Input Port C.

Input Port C transmits data through the CSU to a receiving Input Port (D) in another net node. A flow control signal from Input Port D arrives at Input Port C as the **IfcI** signal.

Appendix B: Design details of the Input and Output Port.

This appendix describes a design of the Input and Output Ports. The first section covers the Input Port while the second describes the Output Port design. In the beginning of each section a brief overview of the Port is given. Then the architecture of the Port is described at a subunit level. Finally some layouts and selected simulation curves are given.

B.1 The Input Port

B.1.1 An overview of the Input Port

The described Input Port supports one-to-one connections. I.e. it does not support broadcast/multicast or interpretation of **advanced switch data fields**.

Main functions.

The Input Port may be divided into two main parts:
• a first-in-first-out (FIFO) buffer, and
• a state machine with a packet interpreter.

The FIFO is used for storing of incoming data when the requested output channel is occupied.

The functions of the state machine and packet interpreter are:
• Creates of an *a4* symbol which together with the first address is forwarded to the crossbar unit. The remaining part of the packet is held back until a positive flow control signal is received.
• Reduces the **jump counter** by one. If the **jump counter** was zero before subtraction the packet will be dismissed.
• Removes the first address nibble and shifts the remaining address nibbles one step forward and inserts the return address nibble at the end of the packet head.
• After a *eop* (end-of-packet) or `Timeout`-signal no other packets are transferred before reconnection has been confirmed by the crossbar unit.

Earlier versions.

In earlier versions the packet was interpreted and manipulated directly at the output of the FIFO before it was handed over to the crossbar. This solution required a state machine with 6 states and 32 transitions (see also Appendix F). It also suffered from long propagation times, and signals had not stabilized within the 10 nanoseconds required for a clockspeed of 100MHz.

Pipeline solution

In the new version the functions performed on each packet have been distributed along a pipeline. The pipeline may be divided into four sections. First an overview of the functions of each of these sections are given.

B.1.2 An overview of the four sections of the pipeline

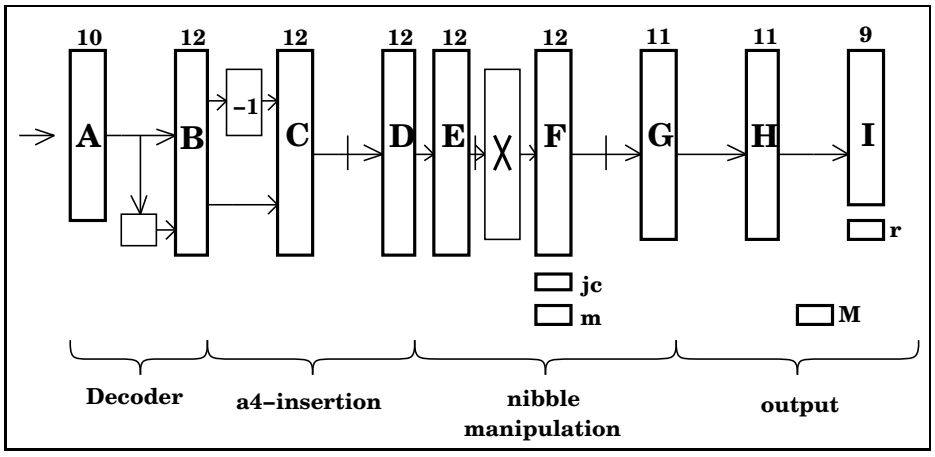


Figure B.1: Sections and registers of the Input Port pipelines.

Decoder section

In this section the incoming 10-bit symbol is decoded. The output symbol will consist of 13 bits where 10 are equal to the incoming 10. In addition 3 bits indicate whether the remaining bits contain a *bop*, *df* or *eop*-symbol or a **jc** equal to 0. The *eop* symbol and the **jc** = 0 condition will be OR'ed together to one signal.

a4 section

This section both subtracts the **jc** by one and creates an *A4*-symbol. The output channel number which is a part of the *a4*-symbol is taken from the lower half of the first word after the *bop*-symbol. Since the construction strategy of the pipeline does not allow new words to be inserted, the *a4*-symbol overwrites the *bop*-symbol.

Nibble manipulation section.

In our protocol address nibbles are removed, shifted and inserted. The most complicated shifting is performed in this section. The output of this section can be loaded in five different ways with parts from one or both of the following words or from a fixed return channel number. A state machine with three states chooses between the five possibilities.

Output section.

The main function of the output section of the pipeline is to start or stop forwarding data depending on received flow control signals. When data are forwarded the output section generates a `shift`-signal which makes all data in the pipeline propagate and new data are read out from the FIFO.

Other functions performed by this section:
• Dismissal of erupted packets.
• If no valid words are available, an *idle* symbol will be transmitted to the crossbar.
• New channel requests will not be sent until reconnection of the previous has been confirmed.
• The *bop*-symbol was lost in the a4 section when an *A4*-signal overwrote the *bop*-signal. In the output section a new *bop*-symbol will be inserted after the *A4*-signal before the remaining part of the packet is forwarded.

B.1.3 Main design strategies of the pipeline

The pipeline is built into a FIFO where data have to be shifted through all the FIFO elements.

All sections are controlled by a common `shift`-signal

All sections are controlled by a common `shift`-signal from the output section. In many cases it would have been easier if each section could alter the `shift`-signal before it was forwarded to the following sections. This could be advantageous if a section had empty words and could initiate a shift-signal for the following sections to fill these empty words. The drawback is that combinatoric logic on the `shift`-line will add delays. Since it is of major importance to keep the `shift`-signal fast it has been chosen not to have logic on the global `shift`-signal. This restriction on the global distribution between sections does not limit the possibility for each section to manipulate this signal locally before it is fed to the single registers.

No words may be inserted into the data stream

Due to the selected flow control etc. empty symbols may be everywhere or nowhere inside and between packets. This possibility and the limitation on the global manipulation of the `shift` -signal results in that no section (except for the one controlling the `shift`-signal) can insert words into the data stream.

Local collecting of valid words

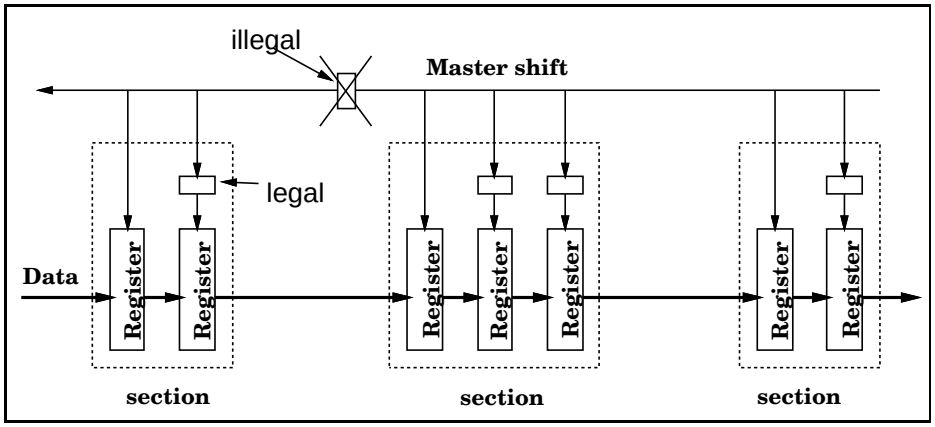


Figure B.2: Globally all sections have to be controlled by the same shift-signal

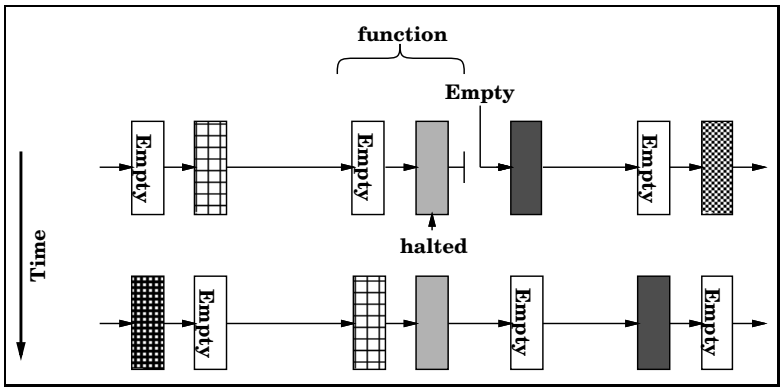


Figure B.3: Collecting data for a function performed on two consecutive words

Some functions are performed on 2 or 3 consecutive valid words. Since empty words may come between valid words we want to hold back valid words until we have the wanted consecutive number of words. Then the function may be performed and the data words may be forwarded. Empty words will pass through to the following sections without this delay.

The number of empty or valid words are not changed. The words only change position.

Figure B.3 illustrates this.

Read on positive clock edges.

All pipeline registers read on the positive clock edge. In the following description it will be default that this shifting takes place on the first positive clock edge after a high global `shift`-signal. If register X shifts according to a more complicated function of `shift` this will be indicated by a $shiftx$ equation.

B.1.4 The decoder section

The A-register is loaded from the FIFO output on every positive clock edge when the global `shift`-signal has been high in the previous period.

$Av := FIFO_{out}v$
$A[0:8] := FIFO_{out}[0:8]$

10 bits of the B-register are loaded directly from the A-register (Bit v and bit $0-8$). $Bbop$ will be high if the loaded symbol is a *bop*-symbol and Bdf will be set high if it contains a *df* symbol. $Beop$ will have a high value either because B contains a *eop* symbol or because it contains a **jump counter** equal to 0.

$Bbop := Av \cdot \overline{A8} \cdot A7 \cdot \overline{A6} \cdot A5$
$Bdf := Av \cdot \overline{A8} \cdot \overline{A7} \cdot A6 \cdot \overline{A5}$
$Beop := Av \cdot \overline{A8} \cdot A7 \cdot \overline{A6} \cdot \overline{A5} + Av \cdot A8 \cdot \overline{A7} \cdot \overline{A6} \cdot \overline{A5} \cdot \overline{A4} \cdot Bbop$
$Bv := Av$
$B[0:8] := A[0:8]$

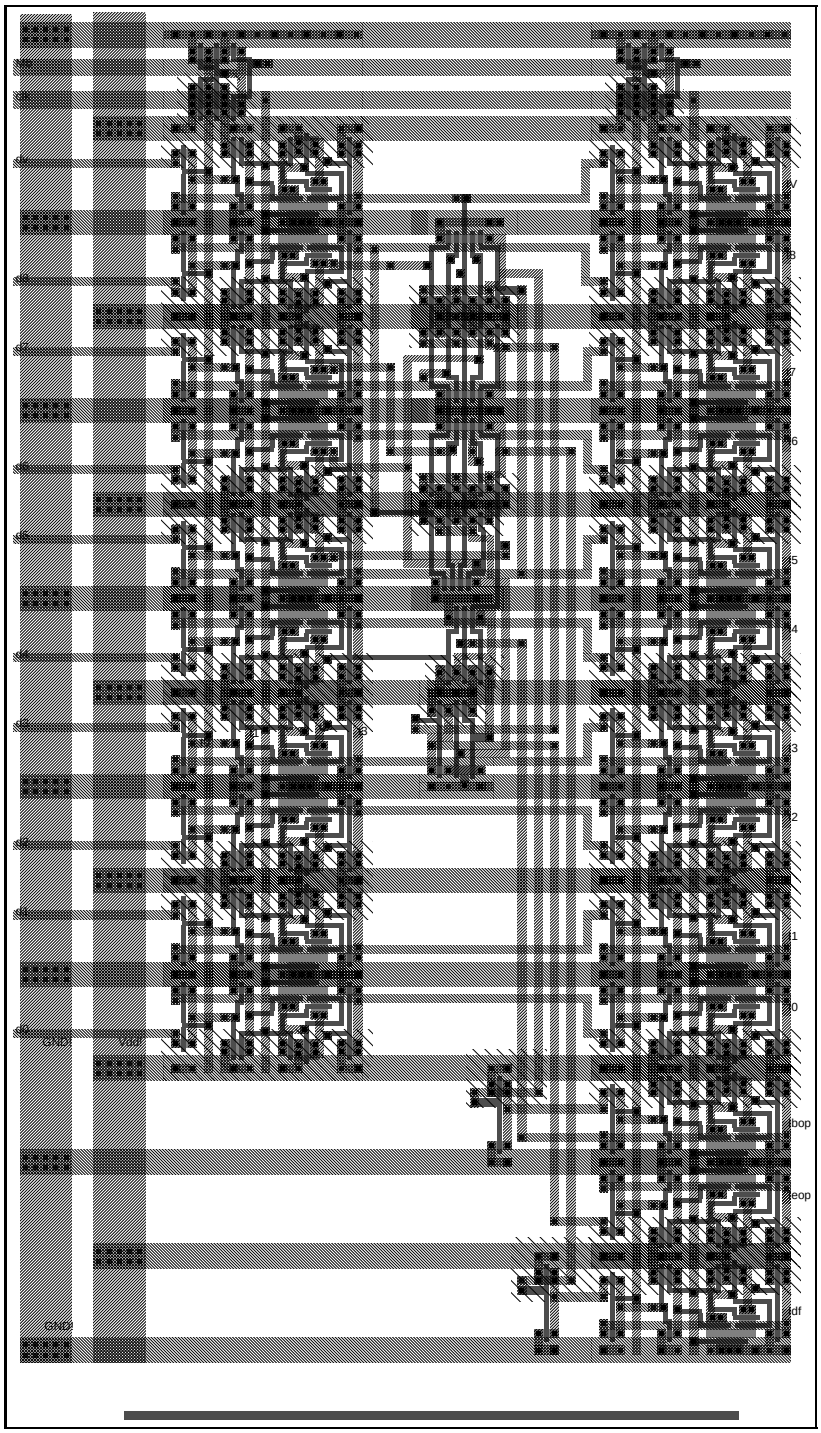


Figure B.4: Decoder section of Input Port pipeline : CIF-layout. The 10-bit A-register is placed on the left side and the 13-bit B-register is on the right side. The decoder logic is in the middle. The sequence of the bits in the B-register are from top and down: Bv, B8 (the data/control bit), the most significant nibble (B7, B6, B5 and B4), the least significant nibble (B3, B2, B1 and B0), the Bbop, Beop and at the bottom Bdf.

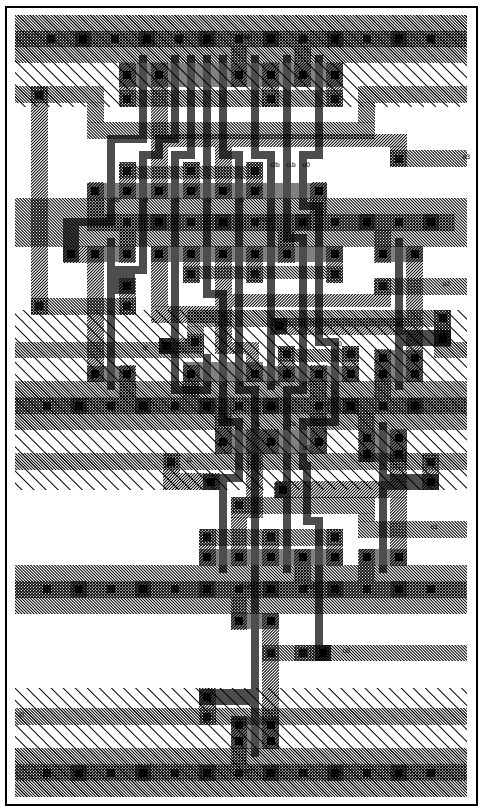


Figure B.5: This figure shows a CIF layout of the subtract-by-one circuit.

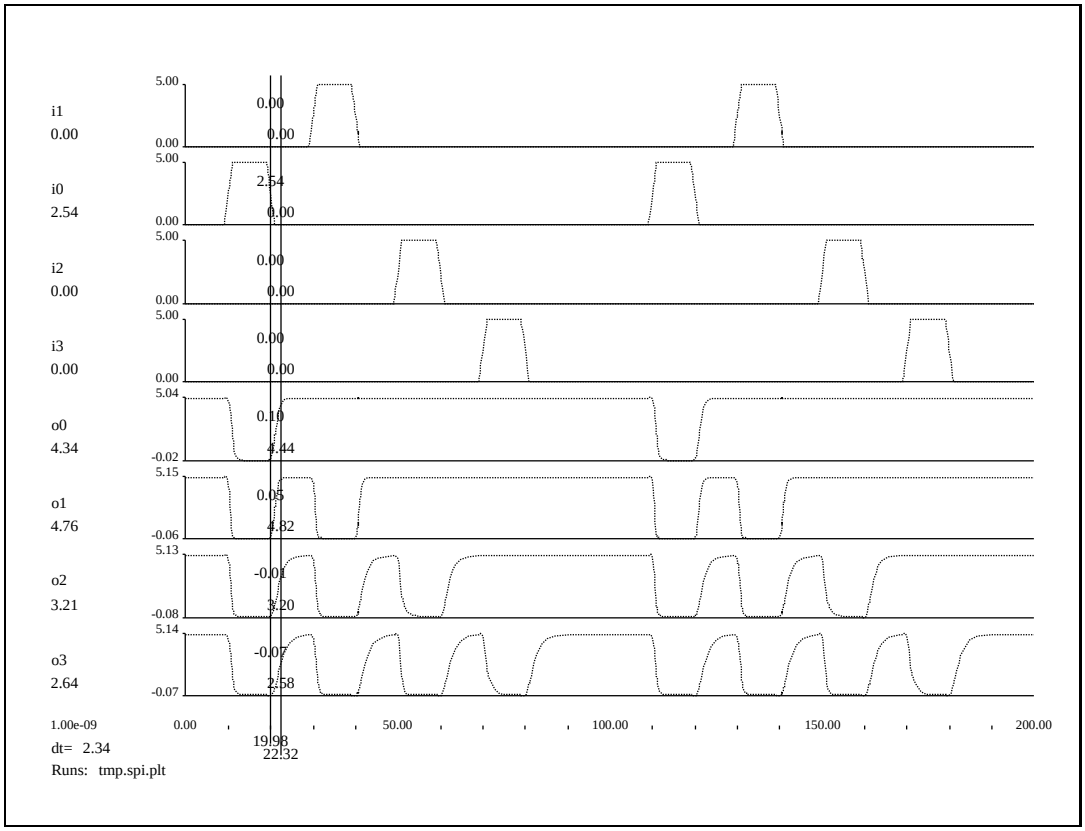


Figure B.6: The figure shows the worst-case behaviour for the subtract-by-one circuit. The simulation has been done with $1.6\mu m$ *process parameters from Orbit using a spice 2gf- simulator. The worst-case delay has been measured to* $2.4ns$.

B.1.5 The a4-section

If the previous word was not a *bop*-symbol the next word to the C-register will be loaded directly from the B-register. If the previous word was a *bop* word the higher nibble of the C-register will load the higher nibble of B (i.e. the **jump counter**) through a subtract-by-one logic.

$Cv := Bv + Cbop \cdot Cv$
$C8 := Cbop \cdot \overline{Bv} \cdot C8 + \overline{Cbop} \cdot B8 + Bv \cdot B8$
$C7 := \overline{Cbop} \cdot B7 + Cbop \cdot Bv \cdot [B7 \cdot (B6 + B5 + B4) + \overline{B7} \cdot \overline{B6} \cdot \overline{B5} \cdot \overline{B4}] + Cbop \cdot Bv \cdot C7$
$C6 := \overline{Cbop} \cdot B6 + Cbop \cdot Bv \cdot [B6 \cdot (B5 + B4) + \overline{B6} \cdot \overline{B5} \cdot \overline{B4}] + Cbop \cdot \overline{Bv} \cdot C6$
$C5 := \overline{Cbop} \cdot B5 + Cbop \cdot Bv \cdot [B5 \cdot B4 + \overline{B5} \cdot \overline{B4}] + Cbop \cdot \overline{Bv} \cdot C5$
$C4 := \overline{Cbop} \cdot B4 + Cbop \cdot Bv \cdot \overline{B4} + Cbop \cdot \overline{Bv} \cdot C4$
$C[3:0] := Cbop \cdot \overline{Bv} \cdot C[3:0] + \overline{Cbop} \cdot B[3:0] + Bv \cdot B[3:0]$
$Cbop := Cbop \cdot \overline{Bv} + Bbop$
$Ceop := Cbop \cdot \overline{Bv} \cdot Ceop + Beop$
$Cdf := Cbop \cdot \overline{Bv} \cdot Cdf + Bdf$

If the C-register does not contain a *bop*, D will copy C. If C contains a *bop* symbol D will load an *a4* symbol instead of the *bop* from C. This can be done when register B has a valid content (i.e. the first address that will make the lower half of the $a4$-symbol). In the waiting time until B has a valid content D will read *idle*s.

$Dv := Cv \cdot (\overline{Cbop} + Bv)$
$D8 := C8$
$D7 := C7 \cdot (\overline{Cbop} + Bv)$
$D6 := \overline{Cbop} \cdot C6 + Cbop \cdot Bv$
$D5 := C5 \cdot \overline{Cbop}$
$D4 := C4$
$D3 := \overline{Cbop} \cdot C3 + Cbop \cdot Bv \cdot B3$
$D2 := \overline{Cbop} \cdot C2 + Cbop \cdot Bv \cdot B2$
$D1 := \overline{Cbop} \cdot C1 + Cbop \cdot Bv \cdot B1$
$D0 := \overline{Cbop} \cdot C0 + Cbop \cdot Bv \cdot B0$
$Da4 := Cbop \cdot Bv$
$Deop := Ceop$
$Ddf := Cdf$

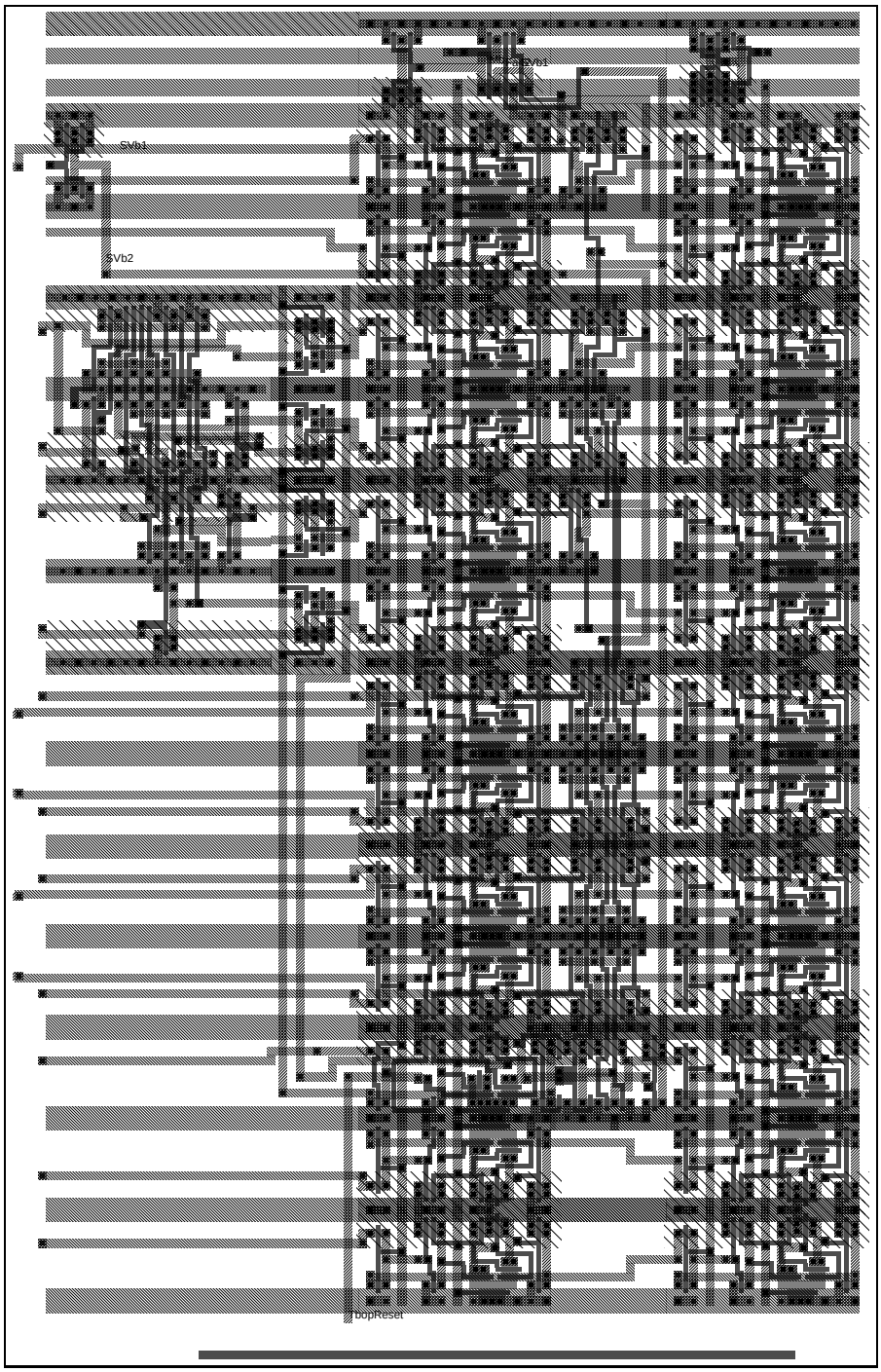


Figure B.7: a4 section of Input Port pipeline: CIF layout. The 13-bit C-register is placed in the middle while the 13-bit D-register is put on the right side

B.1.6 The nibble manipulation section

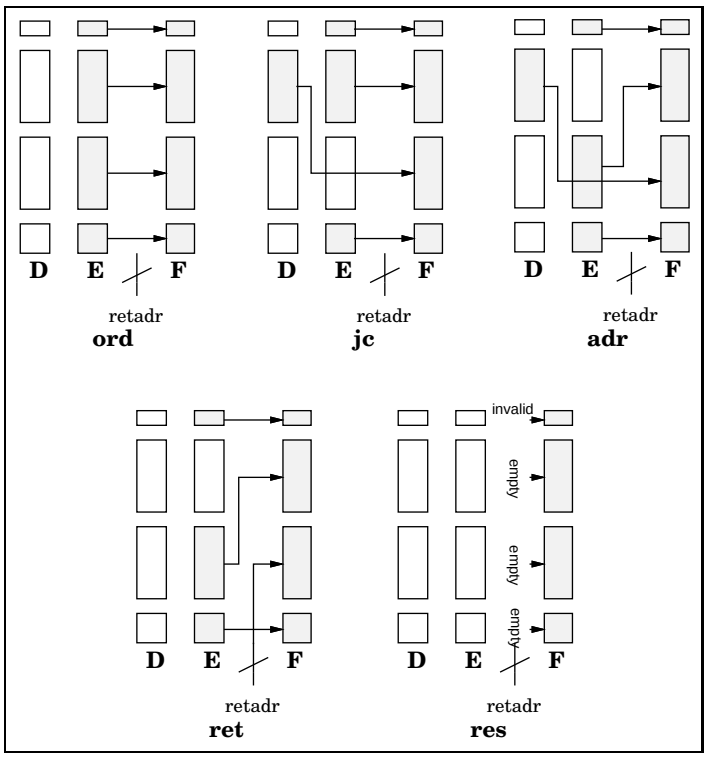


Figure B.8: The main nibble manipulation section of the Input Port pipeline. The figure shows the five different ways in which the ***F****-register can be loaded.*

As shown in figure B.8 the *F*-register can be loaded in five different ways. These ways are named: **ord** (ordinar), **jc** (jump counter), **adr** (address), **ret** (return address) and **res** (reset/empty symbol). The registers are divided into three parts: the high nibble which is bit 4, 5, 6 and 7, the low nibble which is bit 0, 1, 2 and 3 and the control part (*xC*) which is bit 8 and control bits like `xv`, `xa4`, `xeop` and `xdf`.

This section has a small state machine consisting of two bits: `jc` and `m`. `jc` is equal to `Fa4`. Together with two bits from the *D*-register (*Dv* and *Ddf*) they make the basis for how the *F*-register is loaded.

The table below gives the names of the load patterns, their Boolean function, and the register part loaded into the *F*-register.

Name of load pattern	Boolean condition	Register to be loaded into F
`adr+ret`	$m \cdot Dv$	F[7:4]:=E[3:0]
`ord+jc`	$\overline{m}$	F[7:4]:=E[7:4]
`res`	$m \cdot \overline{Dv}$	F[7:4]:=empty
`ord`	$\overline{jc} \cdot \overline{m}$	F[3:0]:=E[3:0]
`ret`	$Ddf \cdot m$	F[3:0]:=retadr[3:0]
`jc+adr`	$jc + Dv \cdot m \cdot \overline{Ddf}$	F[3:0]:=D[7:4]
`res`	$m \cdot \overline{Dv}$	F[3:0]:=empty
`ord+jc+adr+ret`	$\overline{m} + Dv$	F[v,8,a4,eop,df]:=E[v,8,a4,eop,df]
`res`	$m \cdot \overline{Dv}$	F[v,8,a4,eop,df]:=empty

In another form this may be written:
$F[7:4] := E[3:0] \cdot m \cdot Dv + E[7:4] \cdot \overline{m}$
$F[3:0] := E[3:0] \cdot \overline{jc} \cdot \overline{m} + retadr[3:0] \cdot Ddf \cdot m + D[7:4] \cdot (jc + Dv \cdot m \cdot \overline{Ddf})$
$F[v, 8, a4, eop, df] := E[v, 8, a4, eop, df] \cdot (\overline{m} + Dv)$
$shiftE := M \cdot (Dv + \overline{m}(\overline{Fa4} + \overline{Ev}))$
$shiftF := M \cdot (\overline{Fa4} + Ev \cdot Dv)$

The two state machine bits (jc and m) are loaded together with the F-register.
$m := jc + m \cdot \overline{Ddf}$
$jc := Ea4$

Register G reads the F-register except when the F-register contains an $a4$-symbol and the two following words are not valid. In the latter case G will read *idle* symbols until the D and the E-registers have been filled.
$G[\,] := F[\,] \cdot (\overline{Fa4} + Ev \cdot Dv)$

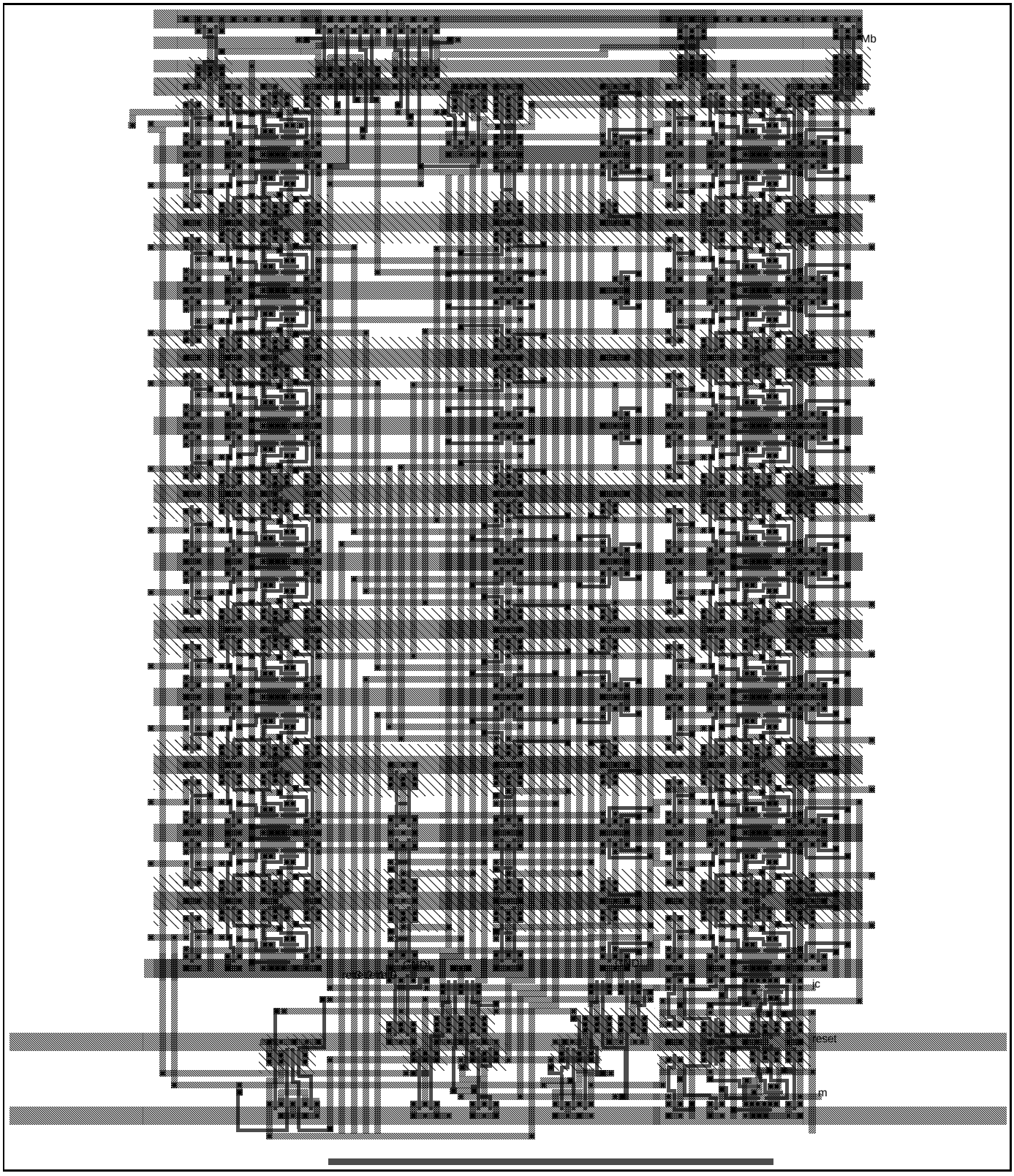


Figure B.9: Nibble manipulation section of Input Port pipeline : CIF-layout. Register E is on the left side while the F-register is on the right side.

B.1.7 The output section

The output section is a state machine with two states. This is implemented by a one-bit memory element: **r**.
0-state is used between packets while
1-state is used during transmission of a packet.
In the 0-state the pipeline contents are shifted until an *a4*-symbol reaches the end of the pipeline. In the 1-state data are forwarded to the crossbar when the incoming flow control symbol (*IfcI* = incoming flow control signal) is high. When either a *eop* symbol reaches the end of the pipeline or a TO (=TimeOut)-signal is received, the state machine will again enter state 0. When the state machine enters state 0 a new packet will not be forwarded until a closure of the previous connection has been confirmed. This is to make sure that a positive flow control signal for the previous packet is not misinterpreted as a flow control signal for the following one.

The equation for the state machine is as follows:
$r := \overline{r} \cdot Ha4 \cdot \overline{IfcI} + (r \cdot \overline{Heop} \cdot \overline{TO})$

The output section forwards to the crossbar either the contents of the **H**-register (ordinary), an *idle*-symbol or an *eop*-symbol. The equations are as follows:
$eop = \overline{r} \cdot IfcI + r \cdot (Heop + TO)$
$ord = \overline{r} \cdot \overline{IfcI} \cdot Ha4 + r \cdot IfcI \cdot qv \cdot \overline{Heop} \cdot \overline{TO}$
$\{idle = \overline{eop} \cdot \overline{ord}\}$

In the a4 section the *bop*-symbol has been overwritten by a *a4*-symbol. This symbol has to be recreated and to be inserted back again into the data queue. After the *a4*-symbol has been forwarded to the crossbar a new *bop* overwrites the old *a4* in the **H**-register. This loading of the **H**-register is decided by:
$lHbop = \overline{r} \cdot Ha4 \cdot \overline{IfcI}$

Another main task of the output section is to decide when data shall be shifted inside the pipeline and out from the Input Port FIFO. It is very important that this mastershift (M)-signal is ready as early as possible after the positive clock edge. To fulfill this the mastershift-signal is evaluated in the previous clock period and stored in a separate one-bit latch **M**. The syntax used in the following is: When x is the output from register **x**, *prex* is the input to the same register.

$M = \overline{prer} \cdot \overline{preHa4} + prer \cdot (preig + \overline{preHv})$

The **H**-register is shifted either on the M-signal or when a *bop*-signal is loaded:
$shiftH = M + lHbop = r \cdot (IfcI + \overline{Hv}) + \overline{r} \cdot (\overline{Ha4} + \overline{IfcI})$

B.1.8 Simulation and layout of the Input Port

The signals in figure B.13 are from top and down: *Reset* for FIFO and pipeline. *TO* is the TimeOut or error signal. Then the flow control signal from the crossbar *IfcI* (incoming flow control signal, *ig* in figure B.13) follows. The return address nibble (*retadr*) is in this case fixed on 9. The clock *clk* speed is 100Mhz. All registers are clocked at the positive edge of the clock-signal. The *IPinpV* and *IPinp* are the data input to the FIFO buffer. *IPinpV* signals whether or not the *IPinp* contains a valid word. The following three rows show the values of some internal signals/buses. *shift* is the common shift-signal of the pipeline (*M*). *q* is the value of the **H**-register while *Hv* signals whether or not this is a valid word. At the bottom we find *IPout* which is the IP output value to the crossbar.

The packet loaded into the Input Port is shown in the *IPinp*-line. The contents of the packet is as follows (in hex):
`0a0`: *bop*,
`132`: **jc**= 3 and first address nibble equal to 2,
`115`: second (1) and third (5) address nibble,
`040`: *df*,
`1fe` and `1dc`: two data bytes and finally
`080`: *eop*.

The packet out from the Input Port is as follows:
`0c2`: *a4*-symbol with first address equal to 2,
`0a0`: *bop*,
`121`: **jc**-decremented to 2, and second address (1) moved forward one nibble,
`159`: third address (5) and inserted return address (9),
`040`: *df*,
`1fe` and `1dc`: two data bytes and
`080`: *eop*.
Notice that the *eop* is repeated until the disconnection has been confirmed (*IfcI* goes down).

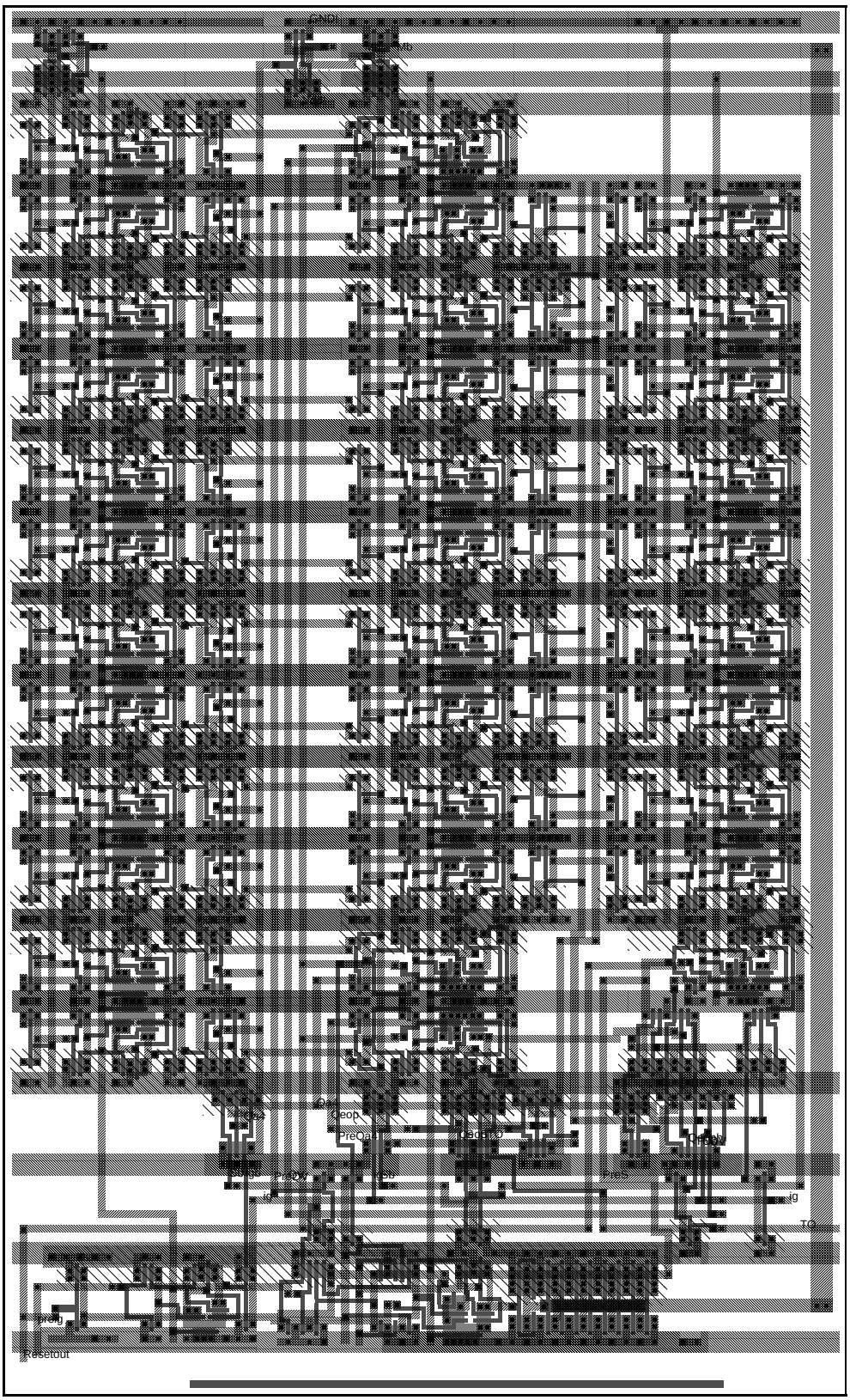


Figure B.10: Output section of Input Port pipeline : CIF-layout. To the left we have the G-register, in the middle the H-register and to the right the I-register. The r-bit is located at the bottom of the I-register. The bottom row shows the M/master shift logic.

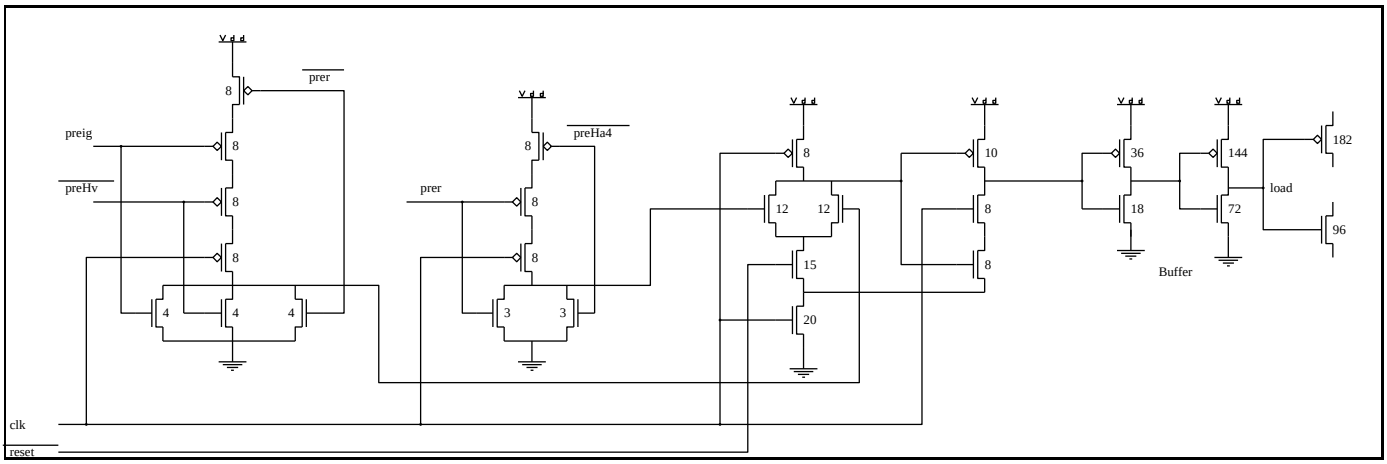


Figure B.11: Circuit diagram for logic generating the common shift-signal of the pipeline. The transistor length is 2λ while the width is given in the figure above. During reset the latch will be set to 1.

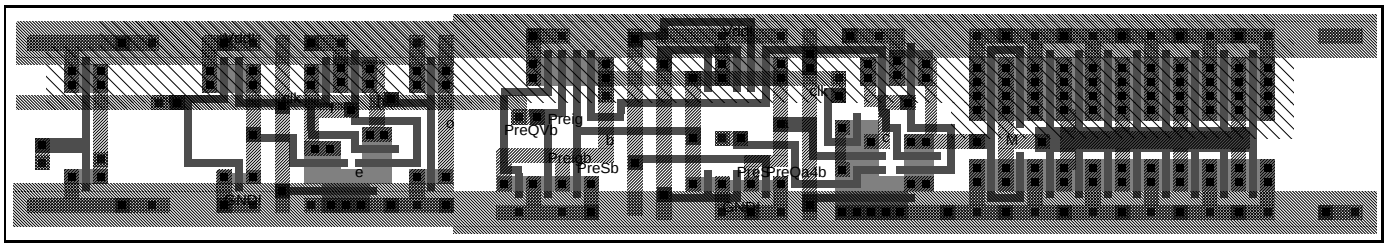


Figure B.12: CIF-layout diagram for logic generating the common shift-signal of the pipeline. The leftmost part shows a memory element for storing of the incoming flow control symbol (IfcI). The middle is the combinatoric latch for the M-function. At the right we find two buffer elements.

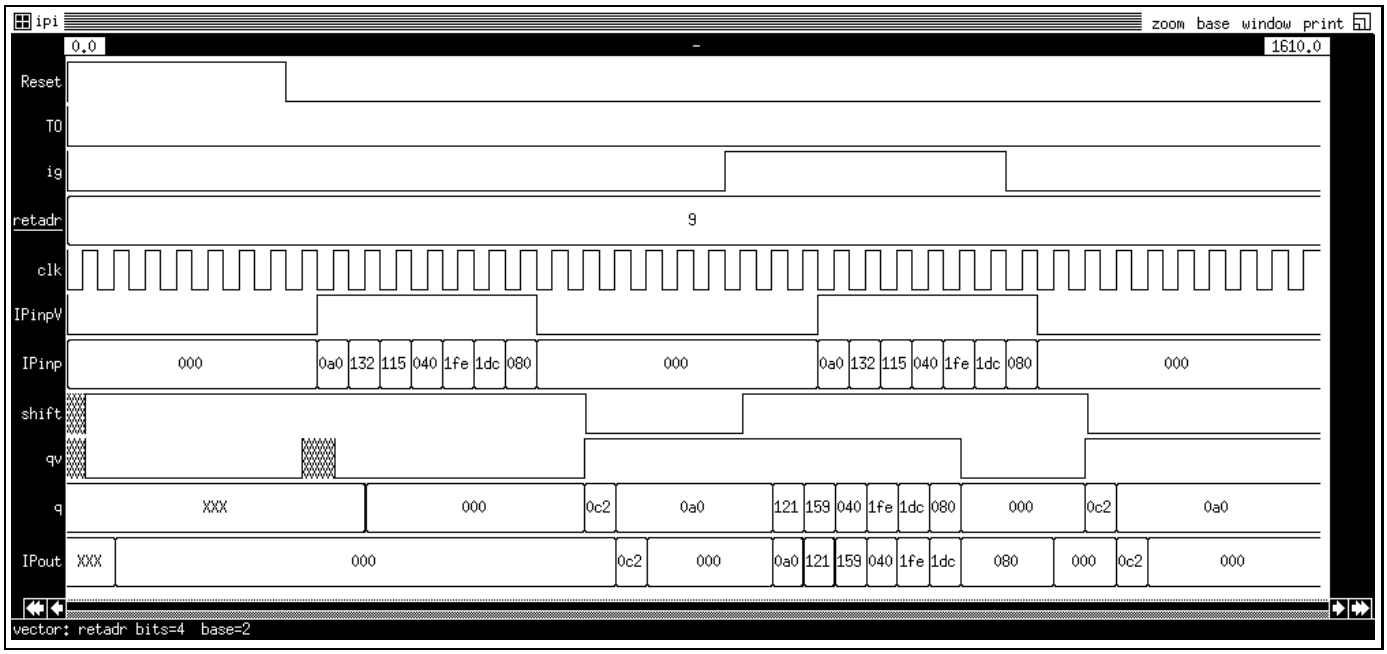


Figure B.13: This figure shows a logical simulation performed on the layout of the Input Port. The `irsim` *simulator uses a simple RC model. The layout is made fast and only small delays can be seen on the simulation curves.*

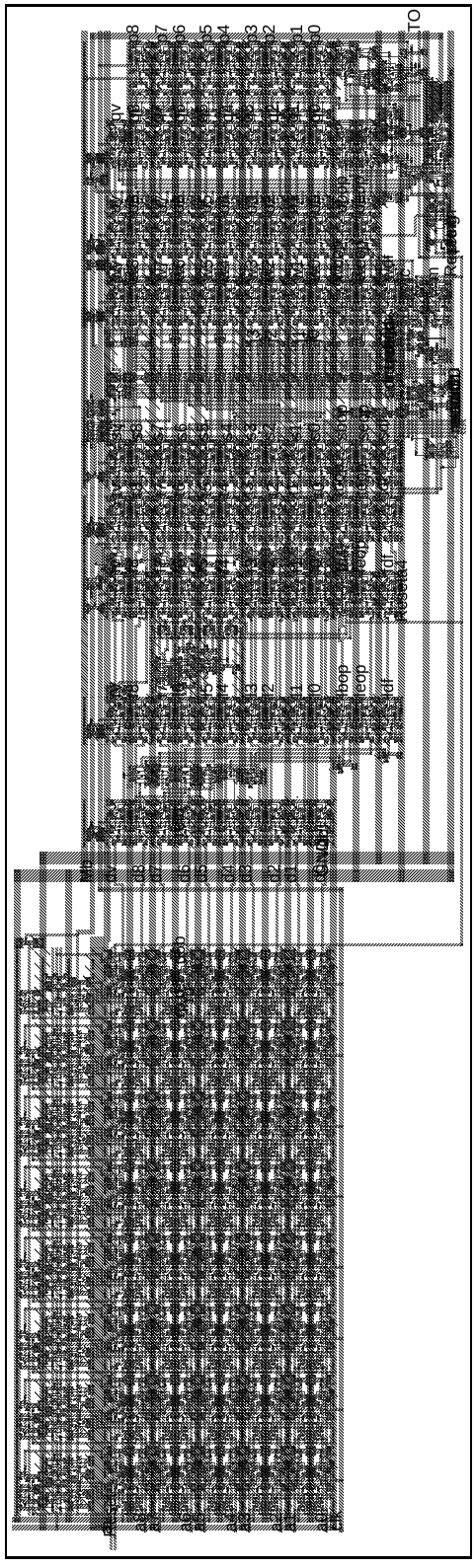


Figure B.14: Layout of the Input Port. The left half shows a 8-word FIFO while the right side shows the Input Port pipeline. Totally the Input Port consists of 3796 transistors. The size of the IP is $3085\lambda \times 919\lambda$*. In a* $1.0\mu m$ *process with* $1\lambda = 0.5\mu m$ *this equals* $1542.5\mu m \times 459.5\mu m = 0.71mm^2$*. Approximately half of this is the pipeline.*

B.2 The Output Port

This section describes the Output Port. First the main functions are defined.

Data out from the CSU pass through the Output Port on their way to the next switch or Protocol Engine. The Output Port has only two tasks: to extract the flow control signal from the data stream as it enters the Output Port and to insert another flow control signal into the stream of data as it leaves the Output Port.

The insertion of flow control signals into the data stream is the most complicated part and most of the logic serves this task. This is because the flow control bit is inserted only in control symbols. If no control symbol are passing a symbol must be created and inserted into the outgoing data stream. Hence a FIFO buffer is needed for the delayed following symbols.

As illustrated in figure B.15 the Output Port can be divided into 6 parts. An overview over their functions is given in the following.

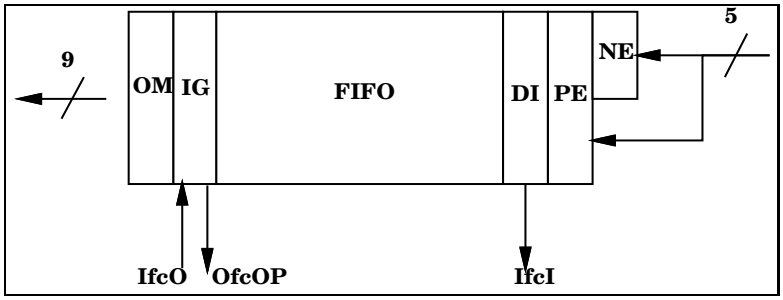


Figure B.15: Block diagram of the Output Port

NE and PE

Data are transported out from the crossbar both on the high and the low level of the clock signal. Thus data must be clocked on both edges of the clock. Closest to the crossbar the Output Port has a section with 5 bits which reads the 5-bus lines on their negative edge (**NE**). Next we have a 10-bit positive edge register which reads both the 5-bus lines and the output of the 5-bit negative edge register (**PE**).

FIFO

Flow control information is transferred inside control symbols. If no control symbols are available new empty (*idle*) symbols have to be inserted. If new symbols have to be inserted the data stream behind has to be delayed and stored. Each word in the FIFO allows a new change of flow control state to be signalled. When empty symbols are received data will be shifted out of the FIFO but not into it. One empty symbol (*idle*) will empty one word in the FIFO.

IG

Still a problem has to be solved: What if the flow control state changes so often and there are so few empty symbols in the data stream that the Output Port FIFO capacity is exceeded ? Flow control signals have to be transmitted in such a way that after a "start transmission" it will always be possible to send a "stop transmission". If this is not the case the "start transmission"-signal can not be transmitted. In this case the "start transmission"-signal will be saved until enough buffer space is available. The **IG** (Insert Guard) handels this.

OM

The last module of the Output Port is the output multiplexer (**OM**). It will insert the flow control signal (*IfcO* = outgoing flow control signal) when a control symbol is to be transmitted. It will also make sure that *idle* symbols are transmitted when no other symbols are ready.

B.2.1 NE: The negative edge register

NE is a 5-bit register which reads the bus from the CSU. Since NE reads on the negative clock edge it will read data transferred during the high clock period.

$NE[4:0] := CBout[4:0]$

B.2.2 PE: The positive edge register

PE is a 10-bit register which reads the NE register and the crossbar bus. Both will be read on the positive clock edge. Since the PE register reads the bus on the positive clock edge it will read data transferred during the low clock period.

$PE[9:5] := NE[4:0]$
$PE[4:0] := CBout[4:0]$

Now PE[9:5] will contain data transferred on the positive clock period while PE[4:0] contains data transferred on the negative clock period.

B.2.3 DI: Detection of Idle symbols.

The previous section (**PE**) sends 10 bits. One of these bits is the incoming flow control symbol which is routed directly to the Input Port. A new bit is created. This is the valid bit indicating whether the incoming symbol is different from the *idle*-symbol.

$FIFO_{in}v := PE[9] + PE[8] + PE[7] + PE[6]$
$FIFO_{in}[8] := PE[9]$
$FIFO_{in}[7] := PE[8]$
$FIFO_{in}[6] := PE[7]$
$FIFO_{in}[5] := PE[6]$
$FIFO_{in}[4] := PE[5]$
$IfcI := PE[4]$

$FIFO_{in}[3:0] := PE[3:0]$

B.2.4 FIFO

The Output Port FIFO is a "fall-through" FIFO. Input data are loaded on a bus passing through all FIFO registers. The empty FIFO space closest to the FIFO output in the next period will be the one which reads the bus. In this way the valid words will be collected closest to the output.

B.2.5 IG: Insert Guard

The function of this circuit is to decide whether the output bus is filled with data from the FIFO or loaded with an *idle*-symbol. It will be loaded with an *idle*-symbol when:
• The buffer is empty so that no valid words are ready, or
• The outgoing flow control signal has changed value and
— the outgoing flow control symbol is "stop" or
— the outgoing flow control symbol is "go" and the FIFO has enough buffer space for a following "stop"

Four one-bit latches are used to implement this function: *ffa*, *ffb*, *ffc* and *fdd*. *fdd* is different from the others in the way that combinatoric logic is a part of the latch. Actually latch *fdd* performs equation *vi* defined in the following pages.

Latch *ffa* copies the outgoing flow control (*IfcO*) (the flow control symbol to be transmitted). Latch *ffb* reads *ffa*. Now a difference in *ffa* and *ffb* indicates that the flow control signal has changed value. This change has to be signaled. *ffa* will have the new flow control value. A high value signals "start transmission" while a low value signals "stop transmission".

$IfcO = 1 \Rightarrow$ start/continue transmission,
$IfcO = 0 \Rightarrow$ stop transmission.

$i : ffa := IfcO$
$ii : ffb := ffa$
$iii : f := ffa \oplus ffb$

A "stop transmission"-signal has to be stored and transmitted at a later time when we have a change in flow control state (f) and a "start transmission" signal ("*ffa*") while the Output Port FIFO is almost full (*OfcOP*). This is taken care of by latch *ffc*. The contents of latch *ffc* will be kept intact as long as there is not space for a possible "stop transmission"-signal (*OfcOP* is high).

$iv : ffc := OfcOP \cdot (ffc + f \cdot ffa)$

Flow control status should be signalled when:
$\overline{OfcOP} \cdot ffc$: We have a stored "start transmission"-signal and enough bufferspace for a possible "stop transmission"-signal.
$f \cdot \overline{OfcOP}$: The status has changed and we have enough buffer space.
$f \cdot \overline{ffa}$: A "stop transmission"-signal can always be transmitted. Thus we have the following equation:

$v : fddii := \overline{OfcOP} \cdot ffc + f \cdot (\overline{OfcOP} + \overline{ffa})$

An *idle* symbol is loaded to the output either when the FIFO is empty or when it has a data word in front and flow control change has to be transmitted (flow control signals can be piggybacked only on control symbols, not on data symbols).

$vi : lidle := \overline{FIFO_{out}v} + FIFO_{out}[8] \cdot fddii$
$vii : shiftFIFOout := \overline{lidle}$

B.2.6 OM: Output Multiplexer

The output multiplexer of the Output Port either loads the output bus from the FIFO or sends an *idle*-symbol. If the *shiftFIFOout* signal is low all outputs (except for O[4]) will be low. This is equal to the *idle*-symbol.

$O[8:5,3:0] := shiftFIFOout \cdot FIFOout[8:5,3:0]$

The output function of line 4 is a little more complex.

When a data word is shifted from the FIFO, O[4] will copy FIFOout[4]. If the word at the FIFO output is a control word or the output is loaded by an *idle*, O[4] will copy the outgoing flow control signal *IfcO*.

$O[4] := shiftFIFOout \cdot FIFOout[4] \cdot FIFOout[8] + (IfcO \cdot (\overline{FIFOout[8]} + \overline{shiftFIFOout}))$

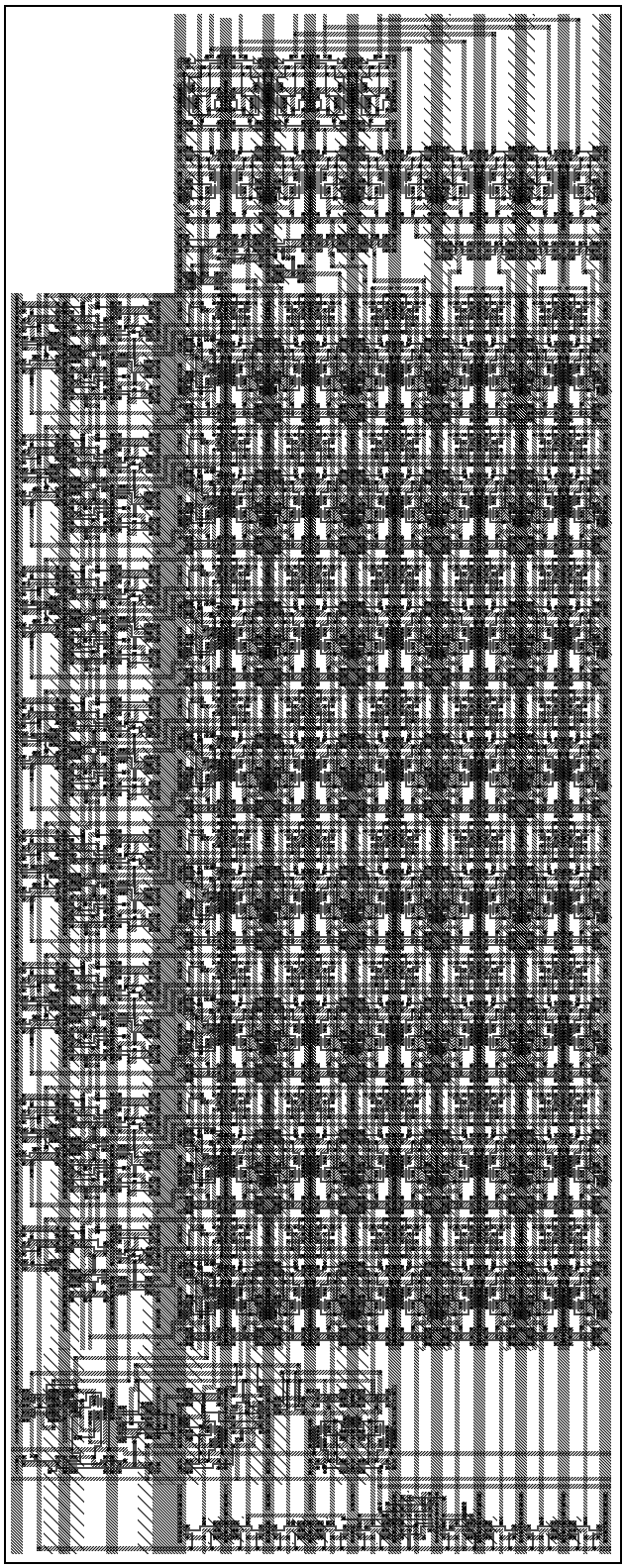


Figure B.16: Cif-layout of the Output Port. The size of the Output Port is $654\lambda \times 1680\lambda$*. With* $1\lambda = 0.5\mu m$ *we have* $327\mu m \times 840\mu m = 0.27mm^2$*. The Output Port has 2 006 mos-transistors.*

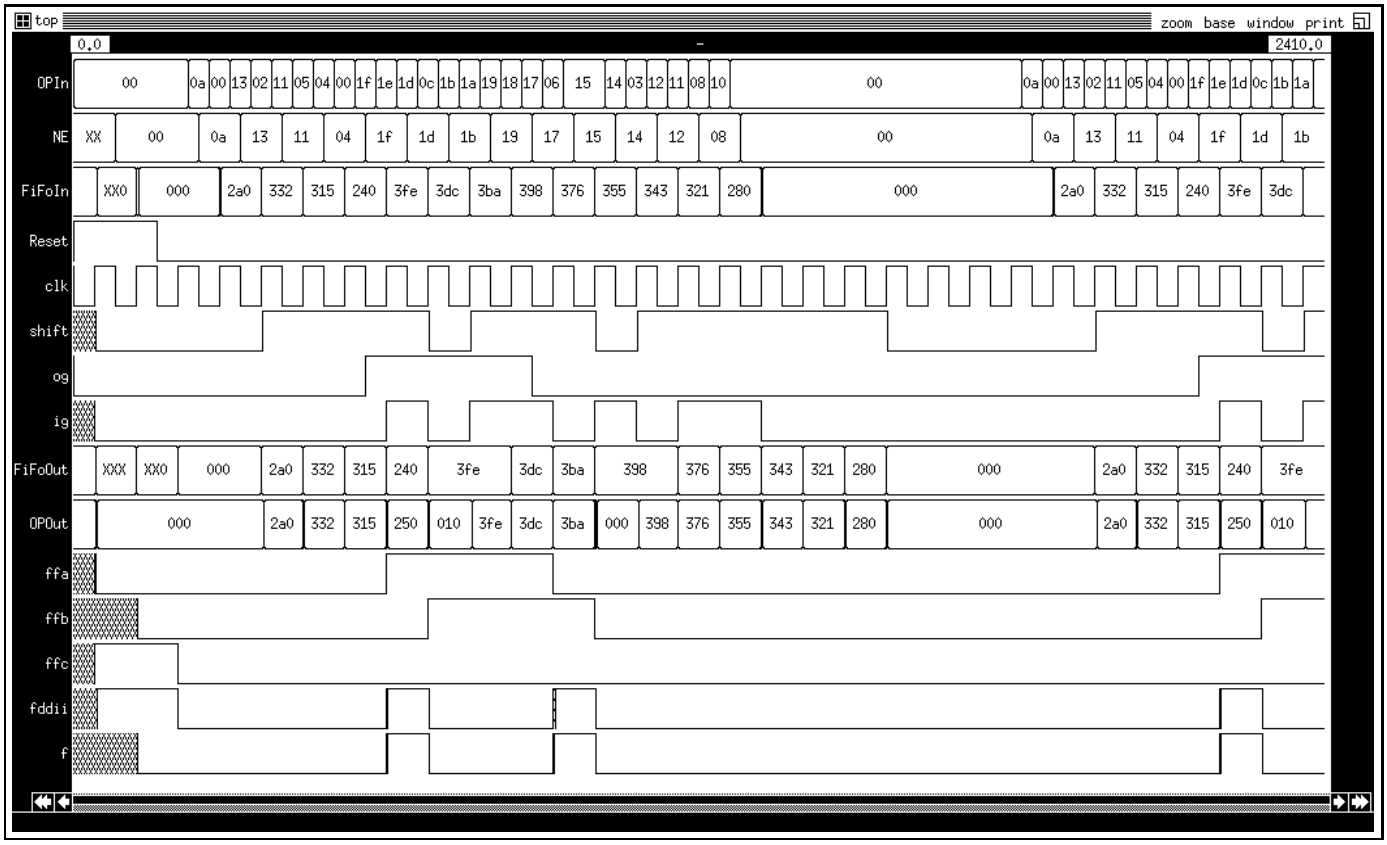


Figure B.17: This figure shows a logical simulation performed on the layout of the Output Port. The `irsim` *simulator uses a simple RC-model for delay analysis.*

The signals in figure B.17 are from top and down: The value on the 5-bit input bus: *OPIn*, two internal buses, the output from the 10-bit negative edge bus, *NE*, and the input to the FIFO: $FIFO_{in}$. *Reset* and *clk* are the global reset and clock signal respectively. *shift* is the local shift-signal. The Output Port output is fed from the buffer when it is high and with an *idle*-symbol when it is low. *IfcO*(*og* in figure B.17) and *IfcI* (*ig* in figure B.17) are the outgoing and incoming flow control signals. *IfcI* arrives from the crossbar while *IfcO* is transmitted on the Output Port output to the following switch or Protocol Engine. $FIFO_{out}$ is the internal output of the FIFO. *OPout* is the 9-bit output bus of the Port. The following 5 curves are internal signals for the decision logic of the output. They are all explained in the subsection about the **IG: Insert Guard**.

The leftmost bit of every second symbol in *OPIn* make the *IfcI* symbol. The remaining part of two and two *OPIn* symbols are put together to one symbol in $FIFO_{In}$. As expected the output sequence of the FIFO is equal to the input sequence. The meaning of the symbols are: $2a0$ is *bop*, 332 and 315 are address symbols, 240 is *df*, then follows some data symbols: $3fe$, $3dc$, ..., 343, 321 and at last: 280 which is the *eop*-symbol. On the line below we see that the Port output is almost equal to the output of the FIFO. The small differences are due to the *IfcO*-signal. It changes value twice during the packet. In the first case an *idle* with the flow control bit sat is inserted (010) while the flow control bit is resat in the second inserted symbol (000). We also see that the control symbols transfer the flow control bit. In the *df*-symbol the flow control bit has been sat so that it changes value from 240 to 250.

Appendix C:
High performance LAN.

This appendixgives utilisation proposals for the SWIPP hardware a little beyond the main goal of the SWIPP approach. In the examples distributed operating system may be running on the SWIPP Protocol Engines.

C.0.1 Examples of utilisation ideas.

The SWIPP hardware and a distributed operating system may be utilised beyond the main SWIPP approach. Examples are given below.
Integrated network: Multimedia applications like video requires high bandwidth. Increased network bandwidth makes it possible to integrate video data with other data traffic in the same network.
Network memory: To avoid disk swapping, clients can borrow memory from servers or other clients in the network for memory demanding applications.
Network cashing: Servers can borrow memory from clients or other servers to increase disk caches.
Parallel filing: The low bandwidth of current disk stations can be increased by dividing the data stream into parallel streams for storage on different disks. Thus the disk bandwidth can be increased up to the connection bandwidth of the network.
Parallel processing: The increased performance of processors results in that workstation systems are more attractive candidates for heavy parallel tasks earlier only performed on dedicated MPP systems. Workstation systems can do this at a lower price and utilise new processors in an earlier stage. Increased network performance reduces the grain size[1], giving a better resource utilisation and a shorter execution time.
Utilising specialised computing resources: A system may consist of both specialised and general computing resources. A global resource management system may queue subprocesses on computing engines according to the subprocess characteristics and the characteristics of the computing resources.
Mixed parallel and serial processing: In local area networks a number of processing elements will always be idle. These can be utilised by allowing parallel tasks to float around on idle processing elements.

All these applications would benefit from increased network performance and global resource management.

[1] "Grain size" is the smallest unit of software for which parallel execution is faster than serial execution.

C.1 The utilisation ideas outlined.

C.1.1 Integrated networks.

In new workstations and PCs traditional data representations like text and drawings are mixed with video, pictures and sound (multimedia). The data sources may be distant.

Applications like these demand higher bandwidth and more predictable transmission time than offered by typical networks of today. Transmission of a $1280 \times 1024 \times 24$ bit frame (full screen) every 1/30 second requires a bandwidth of almost 1 Gbit/sond. Traditional data and pictures can be transmitted on separate nets or they can be integrated. Generally, when resource requests (for network bandwidth) are variable, merging of resources[2] will give lower cost and a better utilisation of the resources.

C.1.2 Network memory.

The execution time of a program will be shortest if the entire program with temporary data can be stored in the main memory. Although memory prices are falling it will seldom be a cost-efficient solution to size the memory for the peak memory request. Thus, in most cases there will be a temporary need for more memory then offered locally. The additional memory can be borrowed in three ways:
- "swapping" to local disk,
- borrowing from memory other places in the network or
- "swapping" to a remote disk in the network.

In client-server systems the remote memory and remote disk will be on the server. "Swapping" means that the least recently used memory pages are stored on disk, making memory space available. The released memory pages may be used for new data or for restoring of earlier stored pages.

Swapping involves a mechanical device, the disk station, which will always have a much higher access time than pure electrical devices. The disk access time is approximately 10^6 times as long as a processor clock period. This relation is not expected to be smaller in the future ([3]). Even when memory pages for swapping are cleverly chosen CPU utilisation may easily shrink to less than 10 % for memory demanding tasks.

	10 Mbit/s Ethernet		155 Mbit/s ATM		622 Mbit/s ATM	
	Remote Memory Paging	Remote Disk Swapping	Remote Memory Paging	Remote Disk Swapping	Remote Memory Paging	Remote Disk Swapping
Mem.Copy	$250\mu s$	$250\mu s$	$250\mu s$	$250\mu s$	$250\mu s$	$250\mu s$
Net Overhead	$400\mu s$	$400\mu s$	$400\mu s$	$400\mu s$	$400\mu s$	$400\mu s$
Net Data	$6250\mu s$	$6250\mu s$	$800\mu s$	$800\mu s$	$200\mu s$	$200\mu s$
Disk	—	$14800\mu s$	—	$14800\mu s$	—	$14800\mu s$
Total	$6900\mu s$	$21700\mu s$	$1450\mu s$	$16250\mu s$	$850\mu s$	$15650\mu s$

The table (based on an idea from [20]) shows the time for remote memory paging and remote disk swapping of 8 Kbytes for some network bandwidths. The relation between remote paging

[2]Replacing two parallel links by one with a bandwidth equal to the sum of the substituted.

and remote swapping increases from 3 for Ethernet ($21700\mu S/6900\mu S$) to 20 for 622 Mbit/s ATM ($15650\mu S/850\mu S$). Reduced network overhead will increase the advantage even further.

C.1.3 Network caching.

Programs are writing and reading to disk stations. Some of the contents written may be accessed later by the same program and some may be read by the next program immediately after program termination. Executables and common tables like font tables are accessed by several programs. Memory copies of what is written to disk will reduce access time for reading. If the the file caches in a network are co-operating, access times will be shortened. The file server will have tables of client cache contents. A file request for contents in a client cache will be forwarded from the server to the cache owner, which will forward the file pages to the the requesting client.

Example

A two-day-trace at Berkeley showed the figures below ([3]). The cluster consisted of 42 workstations each with 16 MB local cache and a 128 MB common server cache.

	Cache Miss Rate	Average Read Response Time
Client-server	16 %	2.8 ms
Co-operative caching	8%	1.6 ms

In the client-server row each workstation has only access to its own cache and to a variable part of the common server cache. The cache hit rate was 84 % giving an average read response time of 2.8 ms. With co-operative caching the cache resources are shared and administrated by the server. This gives a reduction in cache miss rate by a half to 8%. With administration included the new read response time is 1.6 ms.

C.1.4 Parallel file system.

Because of their mechanical parts, disk stations are bottlenecks both when it comes to access time, as discussed above, and bandwidth. Disk access bandwidth ranges from 2 Mbytes/s to 16.6 Mbytes/s for the most expensive new disks.Development of disk stations is expected in the next years to give increased storing capacity while access time and bandwidth are expected not to change significantly until the next century ([53]).

One way to address this issue is the Redundant Arrays of Inexpensive Disks (RAID) systems. RAID systems contain a number of disk stations in parallel. The data stream is divided into several streams for storage on several disk stations. The total bandwidth will be the sum of the bandwidths of each separate disk station. Parallel storage of files may also give better security against disk failures. A data byte and a parity bit may be split into 9 data streams, one to each of 9 disk stations. If one disk crashes data may be regenerated by using the 8 other disks.

Another alternative is to split the data stream already at the user node. The disks accessed can be distributed over the network. With this solution the collected disk bandwidth may only be limited by the network bandwidth. This solution also gives a higher security since the disks are placed on different nodes. With redundancy offering full error corrections any single disk may be removed without loss of information in the network.

C.1.5 Parallel processing.

Massive Parallel Processors (MPPs) are used to reduce the execution time for the largest processes. The processes are divided into subprocesses which are run in parallel by a number of processors. Efficient parallel processing depends both on good processors and on a fast, dense network interchanging data between the processors.

Technological development of integrated circuits has given a high increase in processor performance. Increased performance comes from increased clock rate, increased number of transistors per circuit and increased knowledge about efficient processor architecture. These new processors connected by a high-bandwidth low-latency network offer a performance/price ratio which makes it attractive for tasks earlier only executed by dense parallel machines. The advantage of the workstation/PC networks is their lower cost. Also it has been a tendency that these networks can utilise new processors one to two years before they are used in dedicated specialised parallel processors ([3]). Even though high performing networks of workstations/PCs take over more and more tasks from MPPs they will never take over all tasks. There will always be found new heavy tasks requiring the extra network performance offered by the MPP. In the following an example is given that shows how a high performing network of workstations can compete with dedicated MPPs.

Comparison of some hardware alternatives for GATOR. ([3])

GATOR [25] is a parallel code for simulation of atmospheric chemistry in the Los Angeles Basin. The computation involves input of 3.9 GB, $36 \cdot 10^9$ floating-point operations and output of 51 MB. In [25] three basis systems where compared: A 16-node C-90 from Cray (300 MFlops and 10 MB/s disk bandwidth per CPU), a 256-node Paragon from Intel (12 MFlops and 2 MB/s disk bandwidth per CPU) and different alternatives of a hypothetical network of workstations consisting of 256 RS/6000 (40 MFlops and 2 MB/s disk bandwidth per node). The last-mentioned workstation system is a traditional one with an Ethernet cable connecting all workstations, PVM (Parallel Virtual Machines, a message passing system for workstations from Oak Ridge National Laboratory)[3] communication interface and a traditional sequential file system. The performance of the workstation network is first improved by replacing the Ethernet with an 155 Mbit/s ATM network. In the next step the sequential file system has been replaced with a parallel file system delivering 80% of the aggregate bandwidth of the workstation disks. Finally, PVM has been replaced with a communication system with smaller latency and smaller overhead.

System	Input	ODE	Transport	Total	Cost
16-node C-90 Cray	16	7	4	27	$ 30M
256-node Intel Paragon	10	12	24	46	$ 10M
256-node RS-6000 Workstation network	4030	4	23340	27374	$ 4M
" + 155 Mbit/s ATM	2015	4	192	2211	$ 5M
" + parallel file system	10	4	192	205	$ 5M
" + low latency, low overhead packet protocols	10	4	8	21	$ 5M

The values shown in the table (except in the rightmost column) are time units. The table shows that for the GATOR program a workstation network with all updates would have a shorter execution time (21) than the two MPP systems (27 and 46) at a lower price.

[3]More information about PVM may be achieved through WWW on URL http://www.netlib.org/index.html

C.1.6 Mixing parallel programs with traditional interactive programs.

In a traditional network with a number of workstations/PCs doing interactive work there will always be a number of stations idle. The resources of these computers can be utilised for parallel programs. This has to be done without reducing the performance significantly neither for the user of single workstation/PCs nor for the user of programs executed on several workstations/PCs in parallel.

For simplicity let us say the tasks that are executed in parallel are only running on idle workstations/PCs. We have to know the amount and distribution of idle time to find the total number of stations needed to get a similar execution time to that given if the program was running on reserved stations. Figures based on local network load at Berkeley indicate that a parallel task running alone on N workstations will have a similar execution time if run on $2N$ workstations with a mixture of the parallel task and interactive serial tasks executed for individual users.

C.1.7 User acceptance of resource sharing.

The step from the mainframe to the distributed workstations and PCs gave each user a powerful tool at her/his own table. This is a benefit that users will not easily give away. Thus loaning resources of a machine to another user has to cost so little for the user that it is acceptable. A user being away for some time will probably accept that her/his computer needs a few seconds to be ready for her/him again. After this he/she expects to have the total performance of the computer. In cases where the time to get ready has not been acceptable users have been touching the keyboard frequently to make sure that nobody else takes over their computer. This results in reduced performance since resources otherwise idle are not released.

C.1.8 Summary

The advantages of the ideas presented above are obvious: new programs will start up faster and all programs, large programs in particular, will run faster.

C.2 Main issues to be addressed to utilise increased bandwidth.

C.2.1 Standard Interface software

As far as possible the SWIPP hardware and software should be implemented as procedures with standard interfaces running under common operating systems such as UNIX and DOS. This is to make the system flexible and usable for general application software.

The operating system kernel should be kept unchanged while different processes are changed.

Network read and write procedures.

Network interface procedures have to be simplified. The protocol stack has to be simplified to reduce latency.

The swap daemon

The swap daemon needs a slight change. In stead of storing memory pages on disk the pages are forwarded to a memory somewhere else in the network. The receiving net node needs to have a daemon capable of receiving the page and keeping it in memory. Global resource management processes are needed to watch for unused memory and distribute tables to computing engines in need.

File system

For UNIX there are three dominating file systems: Andrew File System (AFS),Network File System (NFS) from Sun Microsystems and Remote File Sharing (RFS) from AT&T. Of these the NFS is the most used one.

With NFS the **nfsd** daemons run at the disk station while the **biod** daemons run on the computing engines initiating the read/write. Handshaking takes place through remote procedure calls where **rpc.locked** and **rpc.statd** are called on both sides while **rpc.mountd** are running on the side of the initiator. The **nfsd** process has to be changed both to support parellel file storage and to support centralised cache management.

In SWIPP, file system procedures are a part of the global resource management system and are run by the Protocol Engines.

Appendix D:
A 16×16 channel CSU.

This appendix contains a brief description of how a 16×16 *channel CSU can be designed.*

D.0.1 Characteristics and representation.

The design described in this appendix has the following characteristics:
- 16×16 channel CSU (the CSU prototypes discussed in chapter 16 are 4×4 channel switches).
- Monocast only (other CSUs discussed in this thesis support both monocast and multicast).
- Fast unfair arbiter (other CSUs discussed in this thesis use either a fair arbiter or a slower unfair arbiter).
- Data bus width is 5 lines.
- Bit rate on each line is 200Mbps.

Design status:
The schematics on the following pages have been simulated logically at all levels and show correct behaviour. Individual blocks like entrance blocks, arbiters and crossbar elements have also been simulated with analog simulators. Library cells have been designed for an auto-place-and-route generation of the CSU layout. The main purpose has been to get an impression of the size. An auto-place-and-route generated layout can only be used for the lowest clock rates. For higher clock rates routing rules have to be changed, and critical signals, like clock signals, have to be routed by hand. To reduce capacitive coupling the width of the signal lines as well as the distance between them must be increased. For the highest clock rates all lines have to be routed by hand.

D.1 The switch design: Schematics and simulations.

The arbitration logic described by the scheme in D.6 (detail in D.8 and some simulation results in D.7) is a new solution found by the author. This solution is faster (up to eight times) than the architecture used and analysed by the author and several master and undergraduate students during earlier years. The worst-case time to settle a request conflict has been simulated to 5ns. Thus the arbitration task may be done at a clock rate close to the data rate.

Figure D.7 shows a simulation of the maximum delay deciding the clock rate of the arbitration block. The maximum delay takes place when the channels with the highest and the lowest priority initiate a request simultaneously. The delay time is the time from the high priority channel initiates the request until the response from the arbiter to the low priority channel

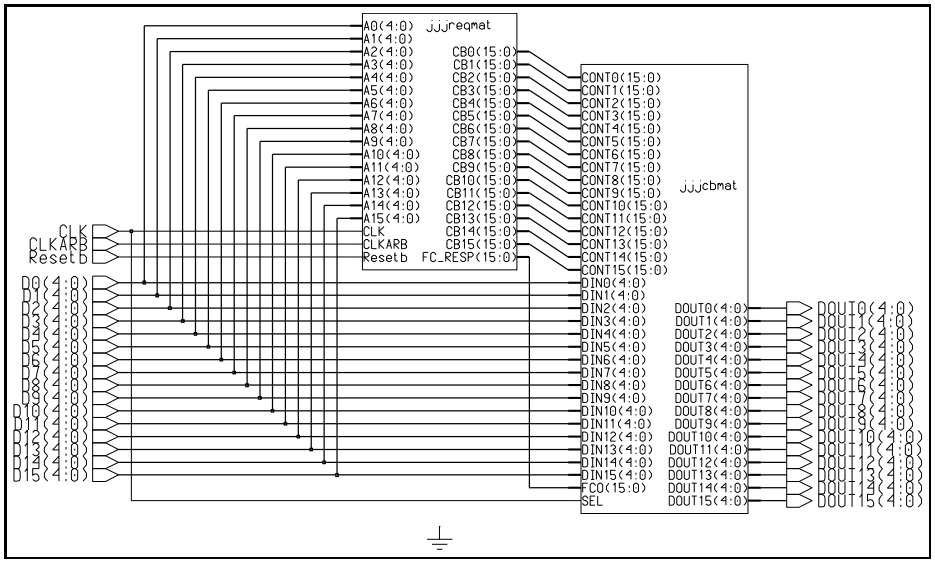


Figure D.1: The figure shows the top level schematic of the switch. The "jjjreqmat" block contains address interpretation, arbiters and request collection. The "jjjcbmat" block contains the switching matrix.

changes accordingly.

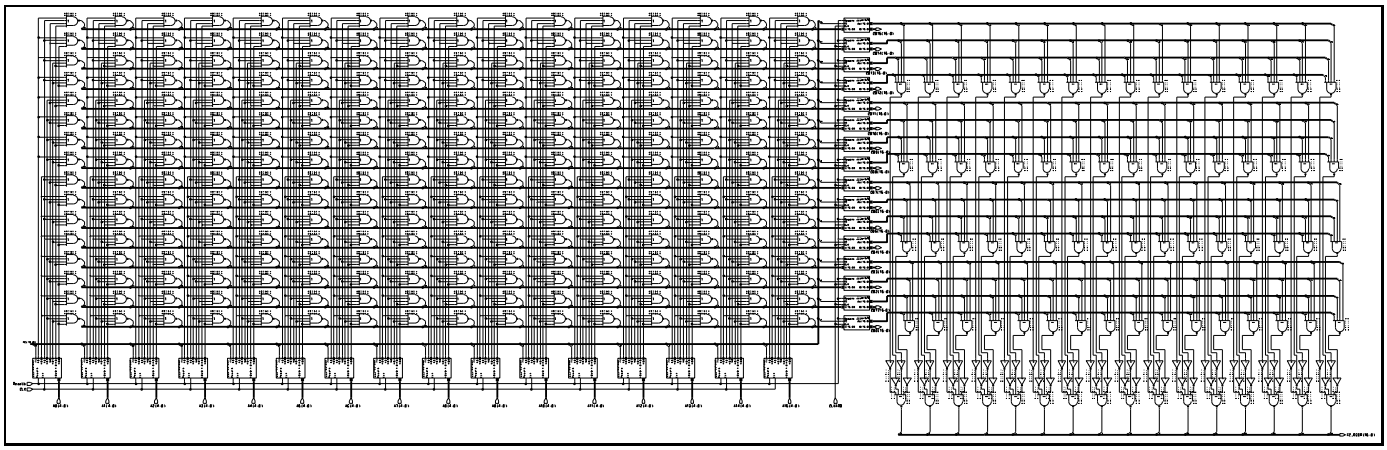


Figure D.2: The 16 rectangular blocks at the bottom on the left side interpret addresses. The 16 rectangular blocks in the middle column are the arbitration blocks. The 16 response signals (IfcI(0:15)), one for each incoming channel, are in the lower part of the right half. The 256 request lines entering the arbitration blocks from the left side are divided into 16 horizontal buses, each with a width of 16 lines. The 256 response lines are on the right side divided into 16 horizontal buses, also 16 lines wide. The entrance block described earlier in the thesis corresponds to one of the address interpreting boxes in the lower left row, a column of sixteen NAND-ports above, and a column of 5 NAND-ports and 4 inverters on the right side of this figure.

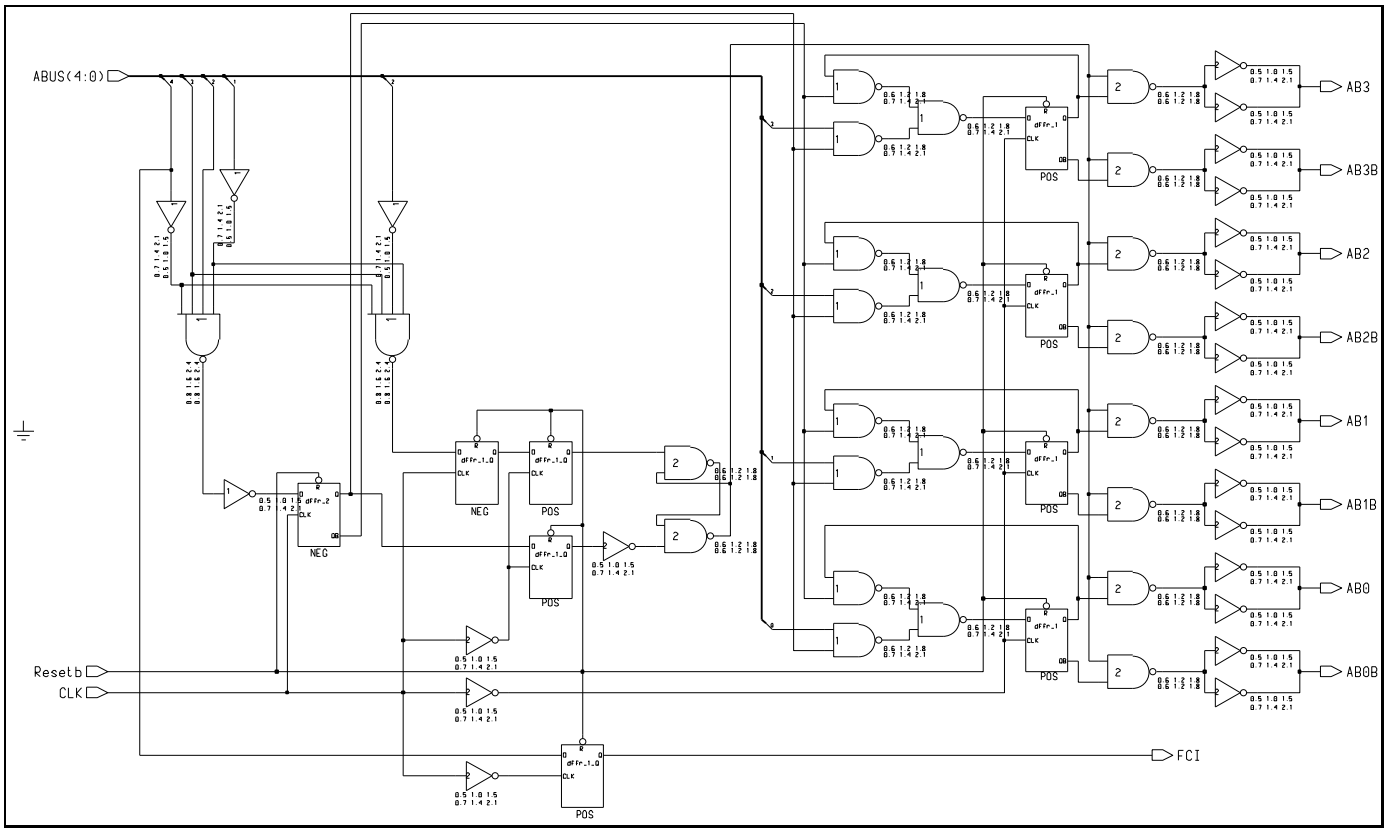


Figure D.3: The figure shows the unicast entrance block. An enable-latch is set by the a4 signal and reset by the eop signal. The identity of the requested output channel is signalled by the four wire pairs on the right side. When requesting a channel, one of the lines in each pair is high. When no outputs are requested all lines are low.

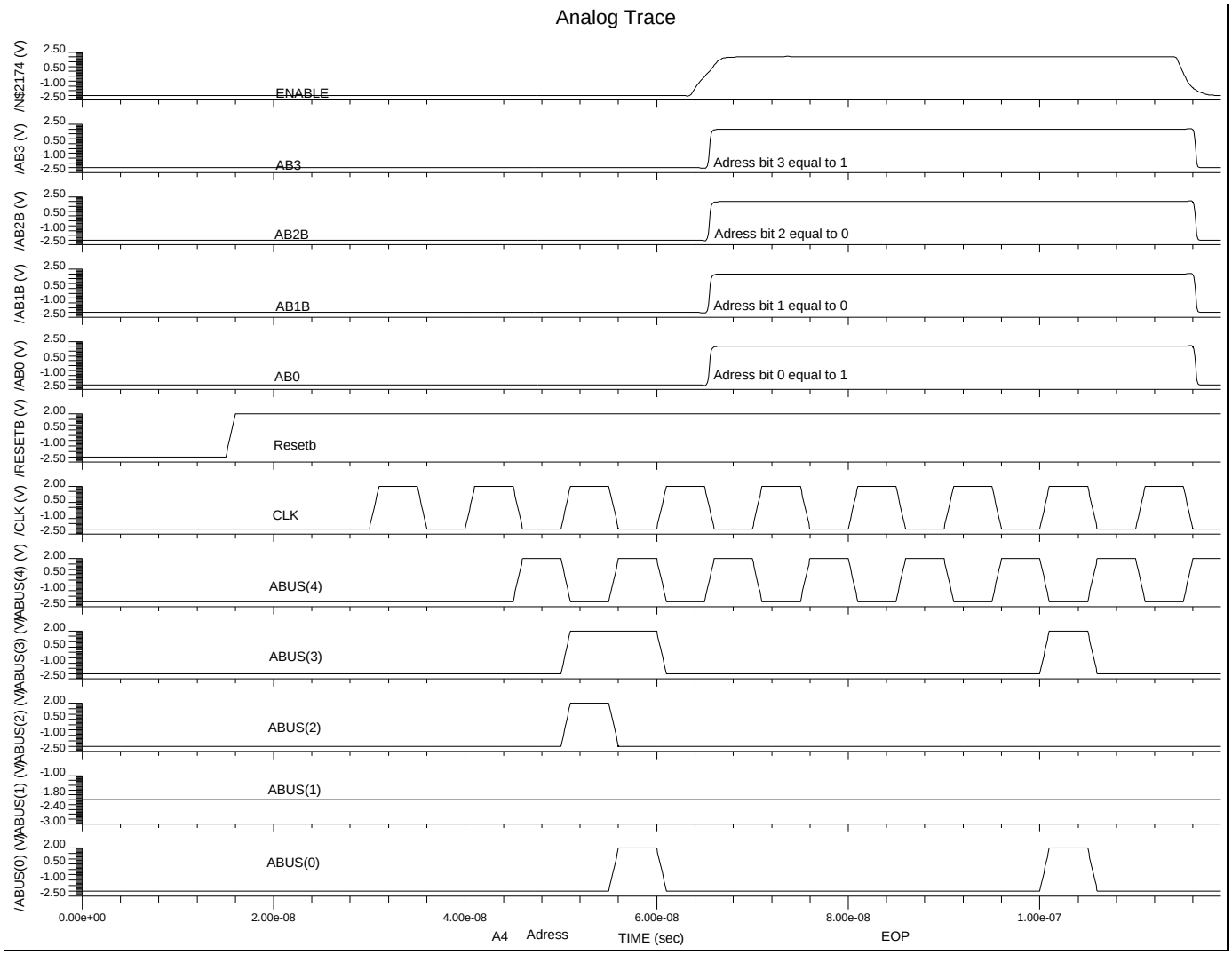


Figure D.4: The simulation shows the loading and resetting of the 4- bit address register in an entrance block. The 5 lowermost signals are the incoming databus. The uppermost signal is the enable signal indicating a valid request address. The four signals below are four of the eight request signals, one from each pair. The clock rate is 100MHz and the bit rate on each line is 200Mbps. The simulation has been done with the slowest latch (the static latch) discussed in appendix J fig. J.1. With these latches the simulations showed correct behaviour when the incoming data signals settled between 0ns and 2ns after each clock edge.

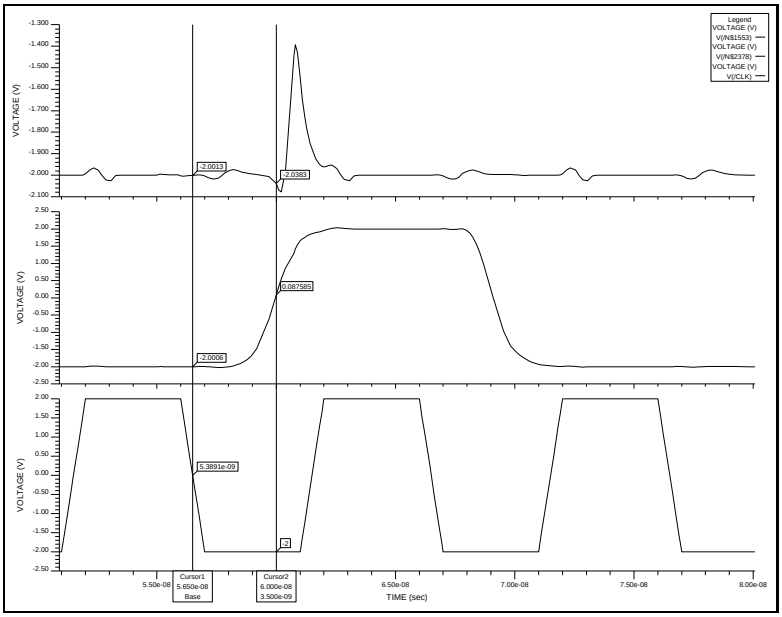


Figure D.5: The main limitation to achieving high clock speed in the entrance block is the slow latches used. The time from a clock edge occurs until the data are valid on the latch output is a very significant part of the high and low clock periods. Clearly, one way to achieve faster clock periods is to choose faster switches.

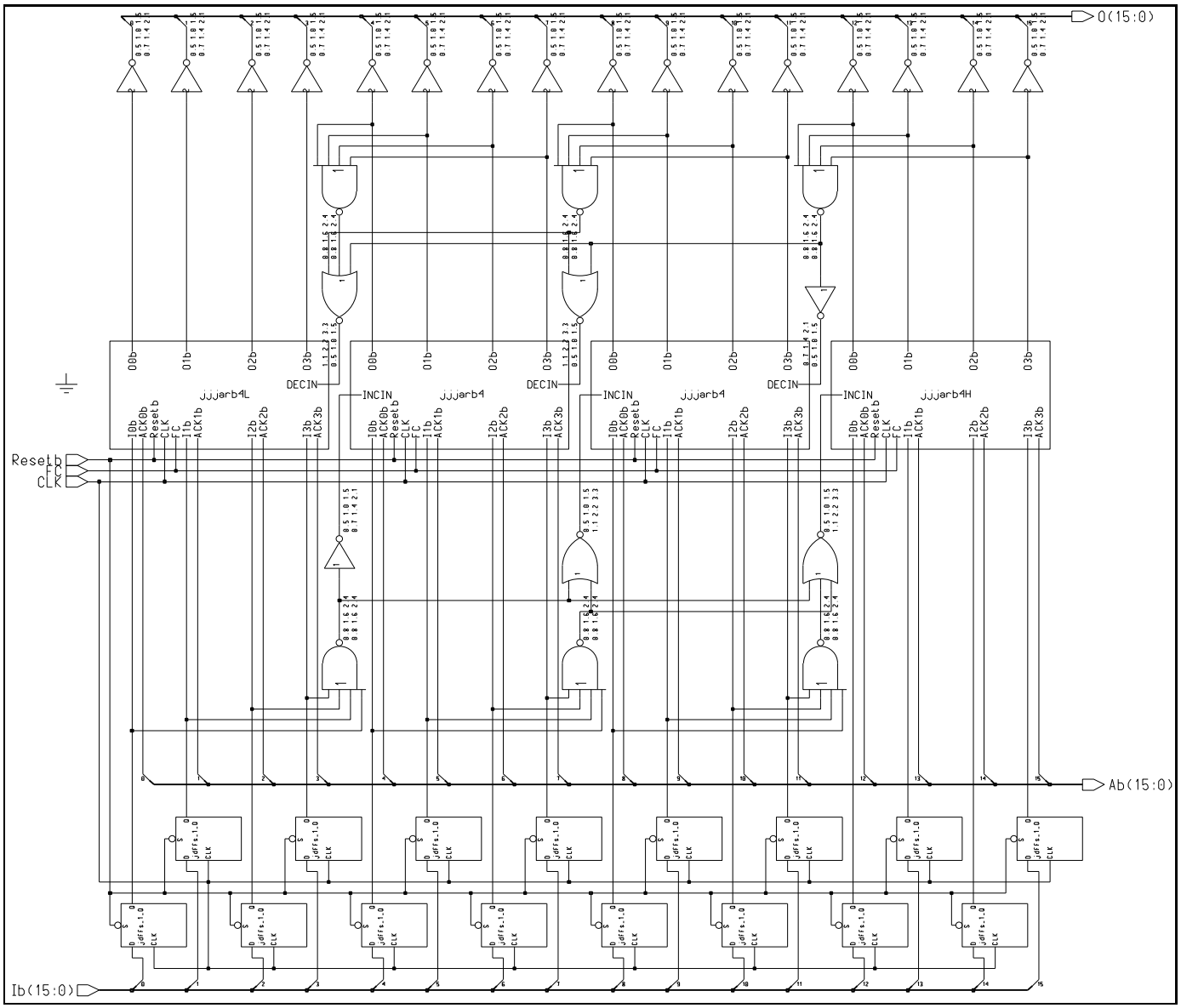


Figure D.6: The schematic shows the arbiter presented at each output channel. The schematic has four blocks, each serving four input channels. The leftmost channels have the highest priority. Each of these blocks may acquire permission to send if neither higher priority channels are requesting to transmit nor lower priority channels are already transmitting. The request signals are latched at the arbiter input by the arbiter clock. The arbiter clock has to be slow enough to allow signals to stabilise after worst-case combinations of requests. If the incoming input channels are allowed to operate on independent clocks also inside the CSU, the arbiter clock period also has to be long enough to minimise meta-stability problems due to requests from different clock-domains.

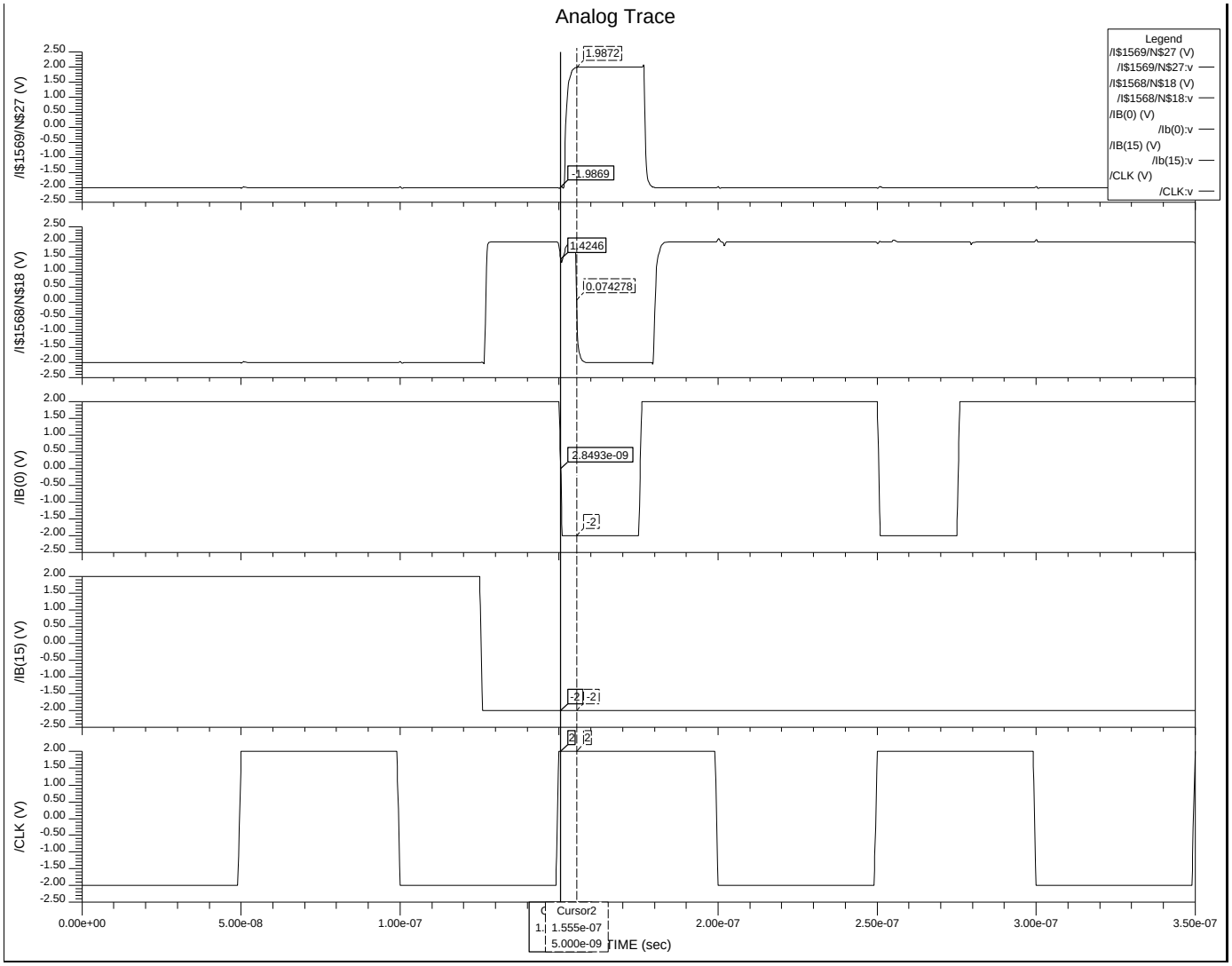


Figure D.7: Simulation of the worst-case delay through the arbitration block. The sequence of the signals are from the top: Response(0), Response(15), Request(0), Request(15) and CLK. Signals are latched at the negative clock edges. The request signal arrivals are a little spread out compared to a real situation. This is done for illustration. In the figure, Response(15) first presents a request and is immediately given a temporary positive acknowledgement. Then Request(0) presents a request. Since channel 0 has higher priority than channel 15, the last channel is losing the temporary positive acknowledgement. This time between the request and the lost acknowledgment is the most critical time interval in the arbitration. It is indicated by the two vertical bars in the figure. A little later Request(0) withdraws the request. At the negative clock edge the arbitration contest is settled, and channel 15 is granted the permission to the output channel for one packet transmission. In the next clock period Request(0) presents a new request. Even though the new request has higher priority it is not evaluated before channel 15 has finished the transmission. The simulation shows that the worst-case delay is 5ns with parameters for a $1.2\mu m$ CMOS process from AMS.

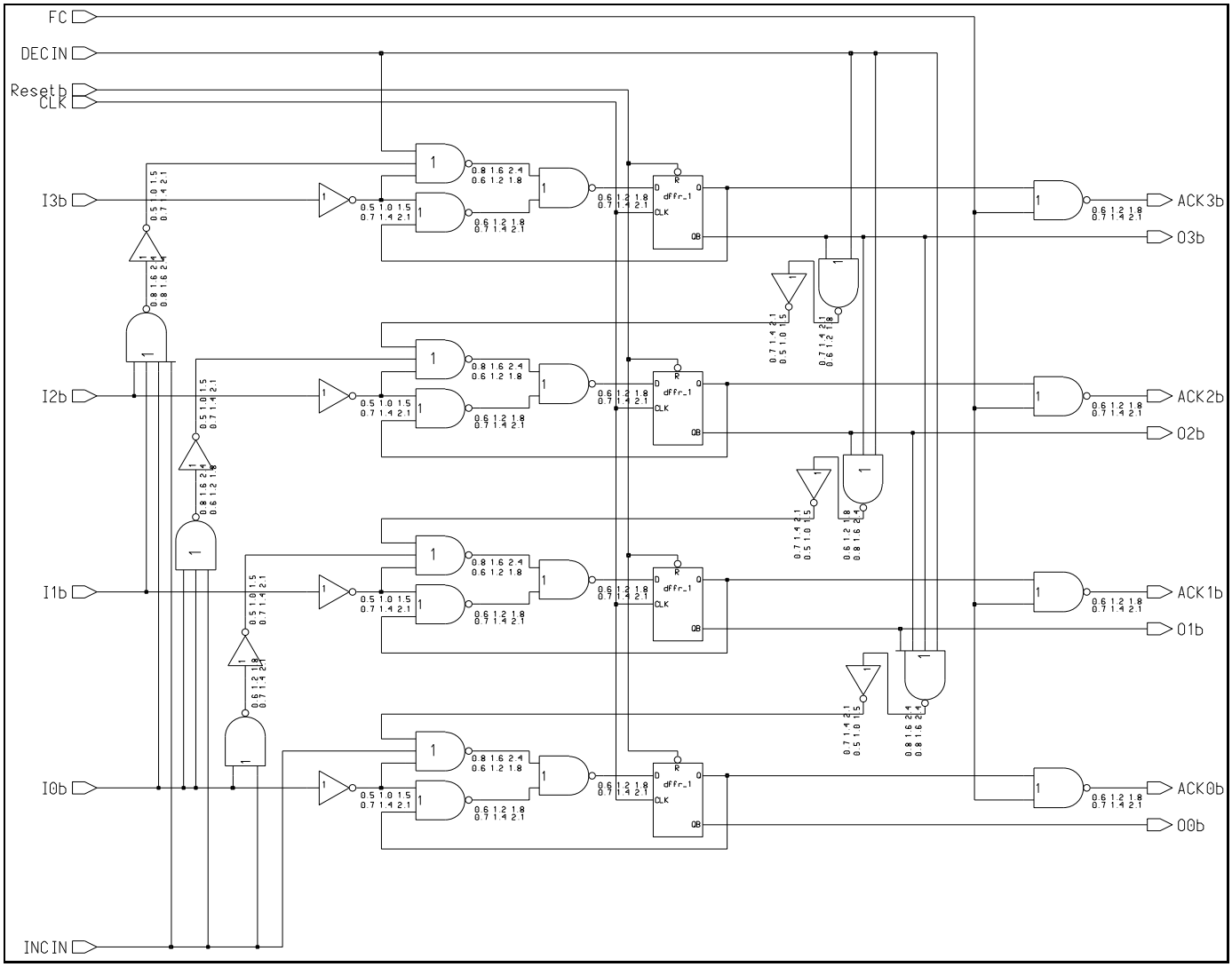


Figure D.8: In the previous schematic the arbiter was drawn with four blocks. The two blocks in the middle are equal. The two blocks on the edges are two slightly simplified versions of the middle blocks. This figure shows the contents of one of the middle blocks. The arbitration algorithm is as described for the unfair arbiter in chapter 13. The arbitration logic of the CSU.
??

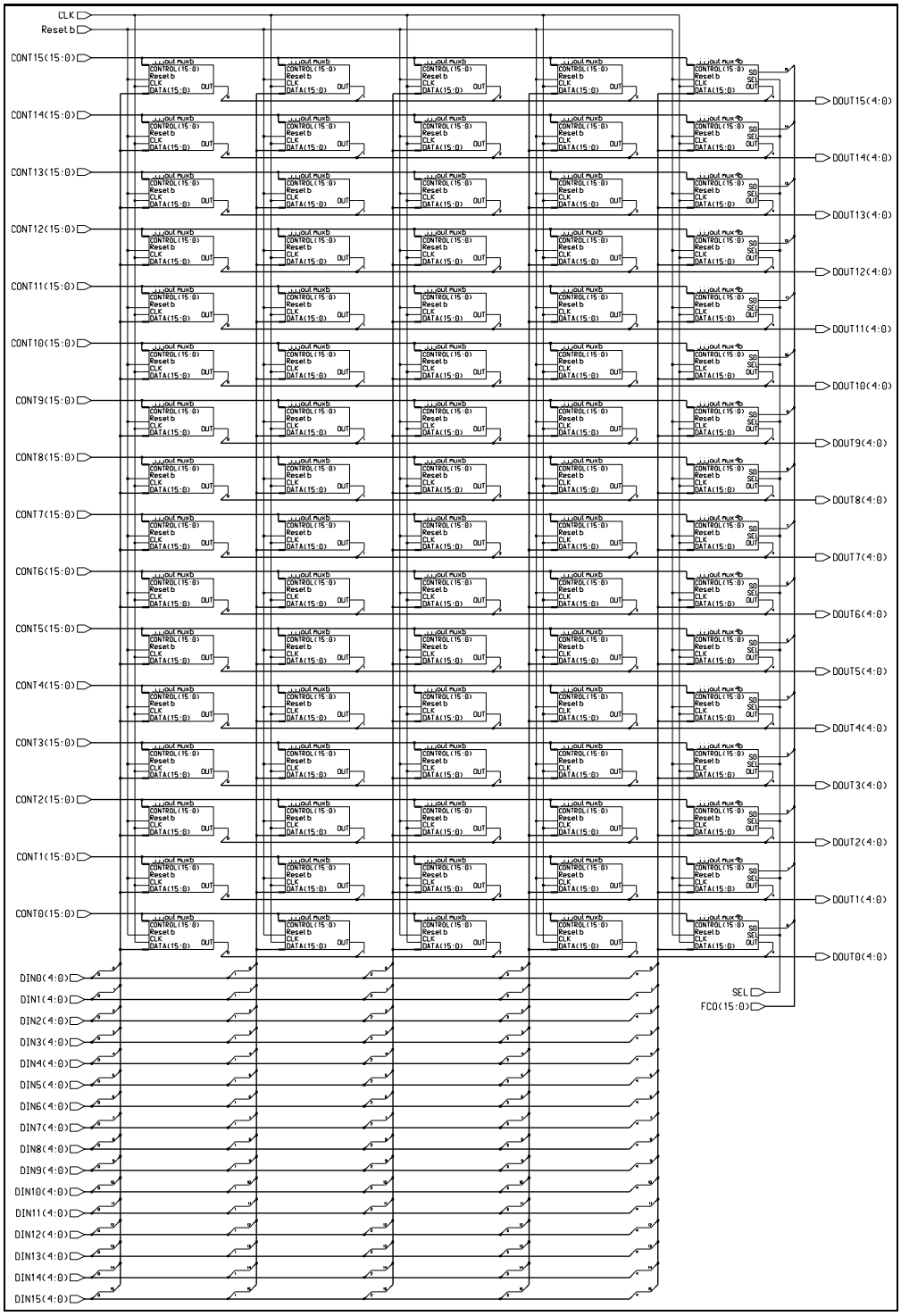


Figure D.9: The schematic shows the crossbar matrix. The incoming data channels are entering in the lower left part. One line from each 5-line-wide incoming channel is routed up in each column. The switching decision lines from each arbiter are drawn from the left side in each column. The outgoing data channel is routed out on the right side. The switching blocks in the four leftmost columns are equal while the right column is a little different, to include the outgoing flow control signal. If the width of the bus word is doubled, five more columns have to be added.

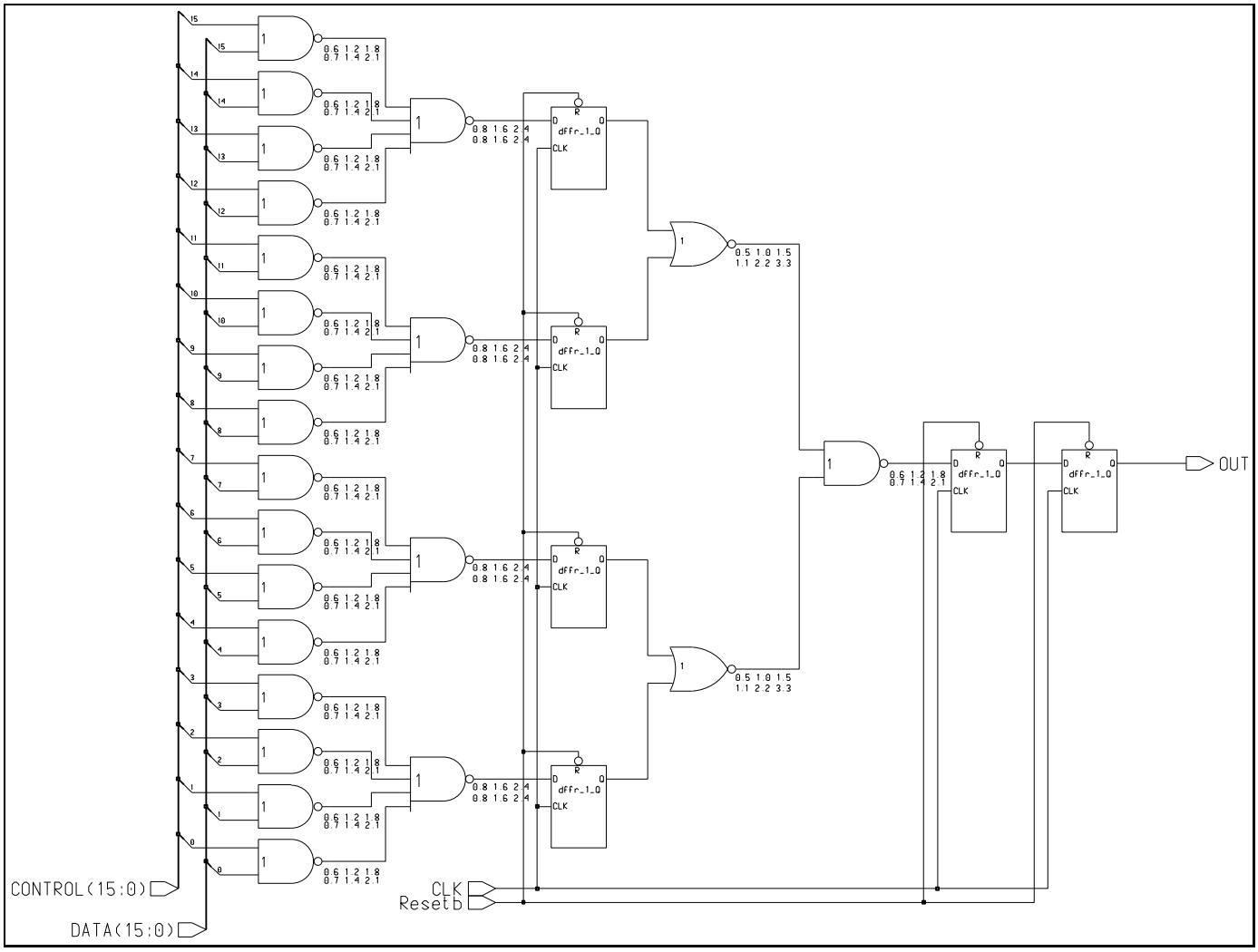


Figure D.10: The figure shows the output multiplexer for each of the four least significant lines of an output channel. Decoders are preferred over transmission gate matrixes and clocked inverters. This is due to advantages in clock speed and noise isolation between input channels. Latches have been inserted to reduce the signal paths between clock periods and thus increase the clock rate. To reduce the clock period further, more latch columns may be inserted to divide the signal paths into smaller parts, or the gate logic may be implemented as a part of the latch, as demonstrated for the Input Port in figure B.11.

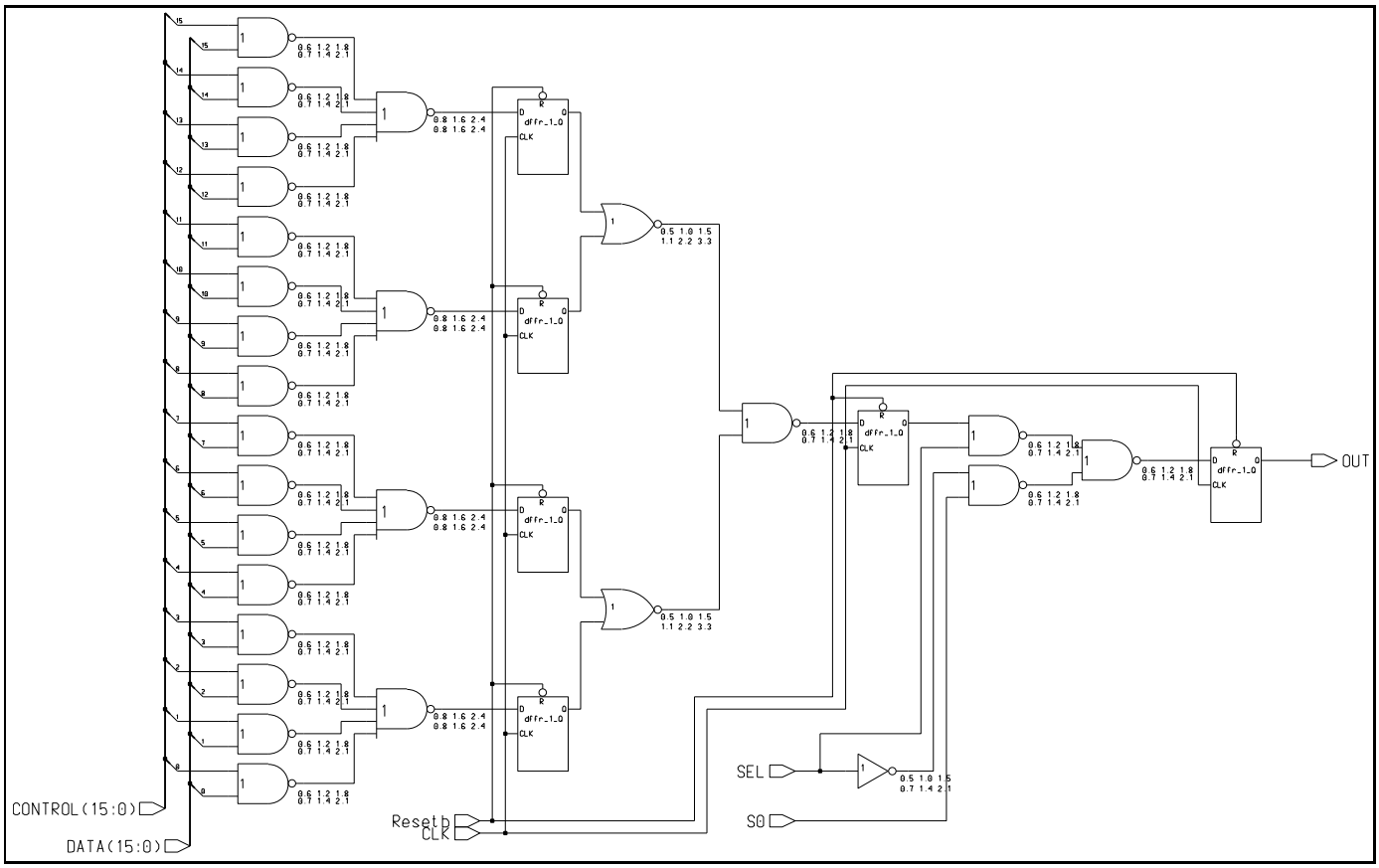


Figure D.11: The figure shows the output multiplexer for the most significant line of an output channel. The extra logic added compared to the previous figure is used to insert the outgoing flow control signal (IfcI) into the outgoing data stream.

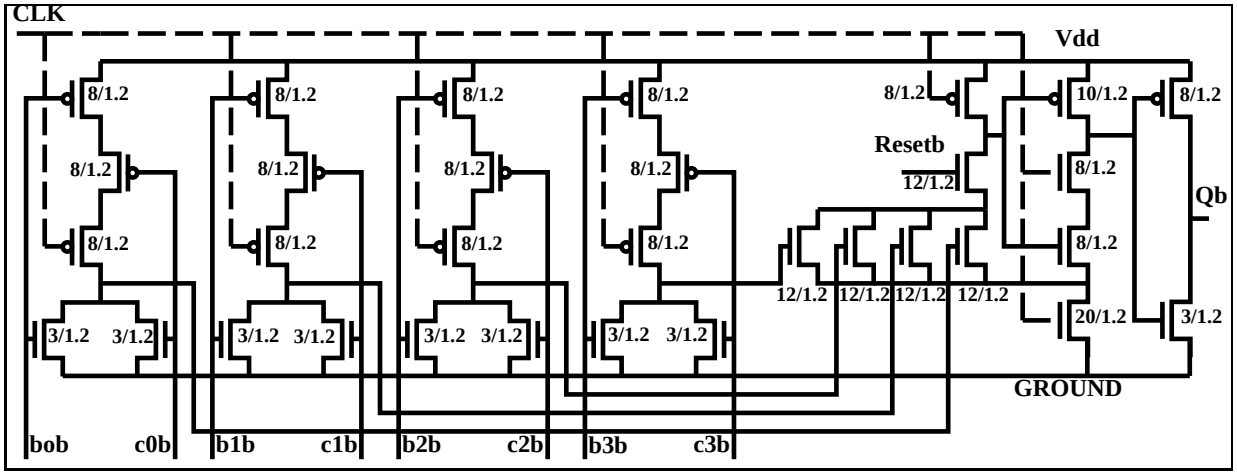


Figure D.12: In CMOS implementations the highest clock rates are achieved with single phase dynamic logic. The figure shows a latched mux for the output multiplexer. The latched mux contains four two-input NOR latches and one four-input NOR port. The inverter at the output may be removed to support higher clock rates. If removed, the data value of the output will be the inverted of the input. Clock rates between 500MHz and 1000MHz have been demonstrated, and higher clock rates are expected as transistor lengths shrink.

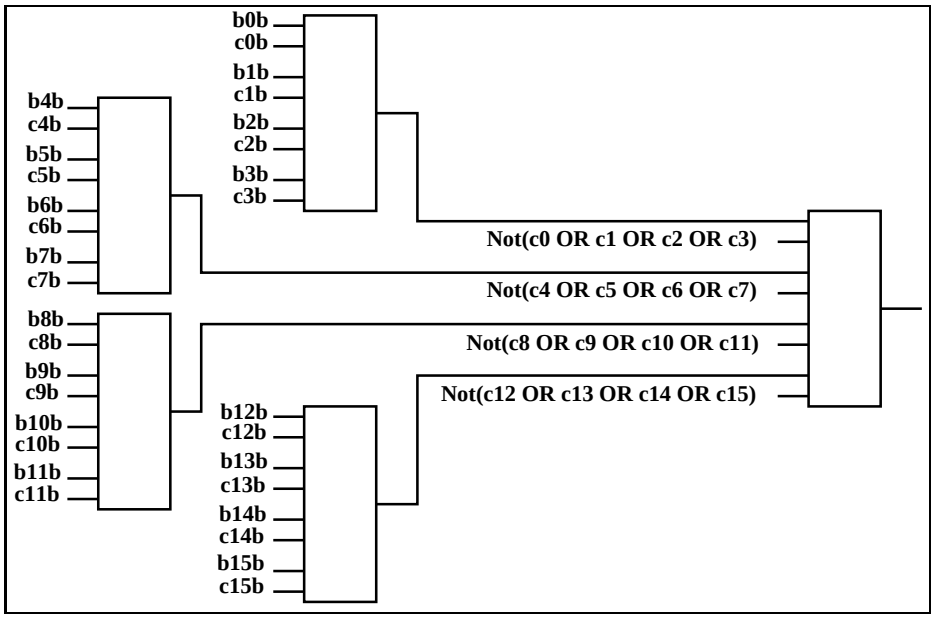


Figure D.13: To be a complete 16 input mux, five of the latched muxes from the previous figure have to be put together as shown in this figure.

Appendix E: Performance of ring, star and N^d-cluster.

E.1 Performance comparison of simple ring and simple star.

For high performance networks the ring and the star topologies are the most popular alternatives. In the following we will give a limited comparison of the performance of the basic structures of these topologies.

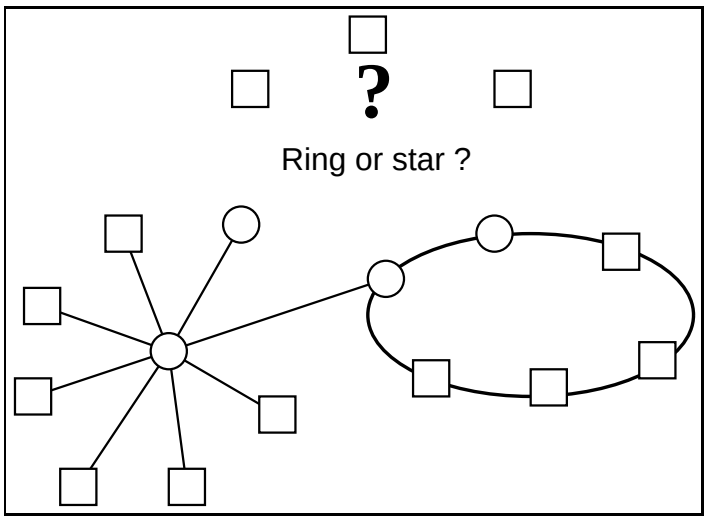


Figure E.1: A network part is implemented either as a ring or as a star. The first part of this appendix gives a limited comparison of these two alternatives, depending on traffic and bandwidth.

We will compare a single ring to a single star. The system consists of N nodes generating and receiving network traffic. The N nodes may be independent computing nodes or bridges to other networks. The following symbols[4] are used in the discussion:

n is the average number of nodes communicating on a segment of the ring. If packets are removed by the destination, n is equal to $N/2$ since an average packet travels half a cycle. If the

[4] The syntax is taken from [61]. Equation E.5 is based on [61] equation 1.59. The remaining part of this appendix is original work by the author. Thus no further references are given.

packet is read by the destination but removed by the source, all packets travel full-cycle and n is equal to N.

λ is the average rate at which a node delivers packets to the network. Thus the average packet delivery rate to a ring segment is $n\lambda$. We choose the probability density distribution $n\lambda e^{-n\lambda t}$ for the intervals t between packet deliveries to a ring segment. We use equal probability distributions for the ring and the star.

C_S is the bandwidth on one star-switch channel in bits per second.

C_R is the bandwidth on the ring channel.

a is the rate between the ring bandwidth and the star-channel bandwidth, i.e. $a = C_R/C_S$. Thus the bandwidth of the ring is aC_S. If a is equal to or less than one, the star topology will always be superior. For our discussion in the following we assume $a > 1$.

$\overline{L_{pk}}$ is the average packet length in bits. The average lengths and the probability distribution of packet lengths are equal for the star and the ring. The probability distribution may be expressed as:

$$\frac{\mu_S}{C_S} e^{-\mu_S \frac{L_{pk}}{C_S}}$$

We use equal packet length distributions for the star and the ring.

ρ_S is the load of the star-channel, i.e. the part of the time a star channel is busy transmitting data. ρ_S is equal to $\lambda\overline{L_{pk}}/C_S$.

ρ_R is the ring load which is equal to $n\lambda\overline{L_{pk}}/C_R$.

ρ_{Ri} is the load contribution on the ring from each node. ρ_{Ri} is equal to $\lambda\overline{L_{pk}}/C_r$.

$\Rightarrow$ Notice that by *traffic* we mean the density of packets (λ) and their lengths ($\overline{L_{pk}}$). Since the load is depending on bandwidth the same traffic will give different load for different bandwidths.

E.1.1 Maximum traffic levels.

A channel constantly busy forwarding data has a load equal to one. This maximum traffic amount may be expressed by the following inequality for the star channel:

$$1 \geq \lambda \cdot \frac{\overline{L_{pk}}}{C_S} = \rho_S \tag{E.1}$$

For the ring this maximum traffic amount may be expressed as:

$$1 \geq n\lambda \frac{\overline{L_{pk}}}{C_R} = \rho_R \tag{E.2}$$

Equation E.2 may be altered to express an a/n relation relative to the star-switch load as follows:

$$\frac{a}{n} \geq \rho_S \tag{E.3}$$

If the ring topology has to manage a traffic equal to the maximum traffic of the star ($\rho_S = 1$), the following relation between a and n has to be fullfilled.

$$a \geq n \tag{E.4}$$

E.1.2 Conditions for low latency

When n is equal to a we have that $\rho_R = \rho_S$. An increase of n and a (still equal !!) will give a reduction in the latency of the ring. This reduction in latency may instead be used either to add more nodes (increase n) or reduce the bandwidth (reduce a). n may also be increased further to utilise available bandwidth. One disadvantage of having $n > a$ is that the maximum traffic potential of the ring would be less than that of the star.

We see from equation E.3 that a has to be equal to or larger than $n\rho_S$ if the ring should be able to handle the traffic. We will now discuss how large a has to be to have a lower average latency than the star.

The model we are using is based on an infinite number of independent sources which are generating packets that will be forwarded through the part of the network that we are studying. The packets from these sources have to pass through the N nodes at the edge of this network part. Hence some queuing and waiting takes place outside the topology we are studying. This waiting time should be equal for the star and the ring. A premise for the equations is that no traffic passing outside disturbs the traffic in the part we are focusing on. This is a simplification, but such traffic is believed to have the same influence on the ring and the star.

The average time to forward a packet through a star channel (the packet-bandwidth time $t_{pk-bw}(S)$) with no queuing is $\overline{L_{pk}}/C_S$. The waiting time $W(S)$ due to queuing is equal to $1/(1-\overline{L_{pk}}/C_S)$. We focus on channel limitations and ignore saturation limitations in certain switch architectures. With these conditions the packet will not experience any queuing time in the switch structure. The packet-bandwidth time in a ring, $t_{pk-bw}(R)$ is $\overline{L_{pk}}/aC_S$. Since n different sources will compete for the same ring, they will suffer a waiting time given by the following equation (based on [61] equation 1.59):

$$W(R) = \frac{n\lambda}{1 - n\lambda\overline{L_{pk}}/(aC_S)}(\frac{\overline{L_{pk}}}{aC_S})^2 \tag{E.5}$$

We want to know under what conditions the average latency of the ring is less than the latency of the star. This may be expressed by the following inequality:

$$t_{pk-bw}(S) + W(S) > t_{pk-bw}(R) + W(R) \tag{E.6}$$

By inserting the expressions and adding on the left and right side we get the following inequality:

$$\frac{1}{1 - \lambda\overline{L_{pk}}/C_S} \cdot \frac{L_{pk}}{C_S} > \frac{1}{1 - n\lambda\overline{L_{pk}}/(aC_S)} \cdot \frac{\overline{L_{pk}}}{aC_S} \tag{E.7}$$

By reorganising equation E.7 we have as follows:

$$a > n\rho_S + 1 - \rho_S \tag{E.8}$$

A derivation of the right side of E.8 on ρ_S gives $n-1$ which tells that the right side is increasing when ρ_S is increasing. The $a > 1$ condition has to be met for all ρ_S. We also know that $\rho_S = 1$ will give the maximum requirement on a. Thus $a > n$ will always satisfy E.8.

By multiplying inequality E.8 by C_S, we have the following expression:

$$C_R > n\lambda\overline{L_{pk}} + C_S - \lambda\overline{L_{pk}} \tag{E.9}$$

E.1.3 An example with SCI and ATM bit rates.

We may visualise the relation above based on standard bit rates. The Scaleable Coherent Interface (SCI) [48] protocol is a protocol for secure coherent updating of shared memory pages spread out on a number of computing nodes. The SCI protocol does not depend on a specific topology. Nevertheless a physical ring protocol has been developed for SCI and is often mentioned in connection with SCI. The SCI ring has 16 parallel data lines each clocked at a clock rate of 500MHz. This gives a total bandwidth of 8000Mbit/sec. The ATM standard by ITU[5] [82] defines packet formats, transmission protocols and bandwidths. The ATM standard may be used both for ring and star topologies. Two ATM standard bit rates are 155.529 Mbit/sec and 622.080 Mbit/sec.

With the SCI bandwidth on the ring compared to the highest ATM bandwidth on each star channel, we have $a = 12.8$. For a ring to be capable of carrying the same traffic as the switch, n has to be less than 12.8. When n is less than 12.8 we also have that the average latency of the ring is shorter than the latency of the star for all load levels. If the ring should be faster only for a traffic corresponding to a star channel load equal to or less than 0.5 ($\rho_S \leq 0.5$), we may increase n up to $n < 24.6$.

E.1.4 Bandwidth relation example based on chip pin-count limitations.

We may perform estimates of the bandwidth relation between a ring and a star based on a limited pin-count consideration. We consider two circuits with the same number of pins for data: N_{Pin}, and with the same clock rate. One is used as a network interface between a computer and a ring topology, while the other is used as a switch in a star switch topology. Since the bit rate per line is equal, the bandwidth is expressed as the pin-number in the following. The ring interface has four ports each with a bandwidth of $N_{Pin}/4$: From ring, to ring, from computer and to computer. The star switch has to share its pin-number between all channels. If we choose the in this case best performing ring protocol, where packets are removed at the destination ($n = N_{Pin}/2$), the bandwidth of each switch channel is $N_{Pin}/(4n)$. With these relations we find the simple relation between a and n given below.

$$a = \frac{N_{Pin}/4}{N_{Pin}/(4n)} = n \tag{E.10}$$

With the pin-count conditions given above the ring and the star have the same saturation level and manage the same amount of traffic.

E1.5 Conclusion of the comparison between ring and star topologies.

When $a < 1$ the star topology will always be faster. When $a \geq n$ the ring will have higher traffic capacity and shorter latency. For little traffic the ring latency will be $1/a$ of the latency of the star. For large traffic levels the latency will be similar.

The number of connections (links) are similar for the ring and star topologies, but the ring links should have significantly higher bandwidth to be competitive. The number of network elements are similar: N for the ring topology and $(N + 1)$ for the switch topology. The ring topology requires one type of element while the star-switch requires two.

[5] The International Telecommunication Union, former CCITT

To handle all traffic amounts handled by the star, the ring has to satisfy the following inequality:

$$a \geq n \tag{E.11}$$

If the maximum traffic amount is limited to a certain level giving a star load ρ_S, we may reduce the requirement on a to:

$$a \geq n\rho_S \tag{E.12}$$

The condition when the ring has shorter latency than the star is given by the following equation:

$$a > n\rho_S + 1 - \rho_S \tag{E.13}$$

The time to transmit a packet through the ring, $t_{pk-bw}(R) + t_{wait}(R)$ relative to the time to transmit a packet on the star, $t_{pk-bw}(S) + t_{wait}(S)$ may be expressed as:

$$\frac{t_{pk-bw}(R) + t_{wait}(R)}{t_{pk-bw}(S) + t_{wait}(S)} = \frac{n\rho_S + 1 - \rho_S}{a} = \frac{n\lambda\overline{L_{pk}} + C_S - \lambda\overline{L_{pk}}}{C_R} \tag{E.14}$$

The star switch will have lower latency than the ring under the following condition:

$$n\rho_S + 1 - \rho_S > a > n\rho_S \tag{E.15}$$

The main conclusion is that the ring requires a much higher bandwidth than the star to be capable of managing the same load levels. When the need for bandwidth has been fulfilled the ring offers a better utilisation of the available bandwidth.

E.2 N^d switch clusters.

The performance of the N^d switch cluster class of topologies is interesting not only for the topology itself but also because it is believed to give an indication of the characteristics for irregular topologies. Possible conclusions for irregular structures are discussed at the end of this section.

The switch cluster.

The cluster consists of N^d switches with N switches in each of d dimensions as shown in figure E.2 and E.3. A switch has $2d$ link directions i.e. each dimension has two directions. In each direction the links are connected either to a switch or to a computing node. In this topology the computing nodes are connected on the outside of the switch cluster. Thus the switches inside the cluster are not connected to computing nodes while the switches on the "surface" are connected to between one and d computing nodes.

Traffic model and load calculations.

Each computing node is expected to generate a packet stream of λ packets on average per time unit. Each computing node has an equal probability for transmitting to all computing nodes. The total bandwidth of each switch is pin and clock rate limited. Hence all switches have the same total bandwidth. Some differences in performance are caused by division of the total bandwidth

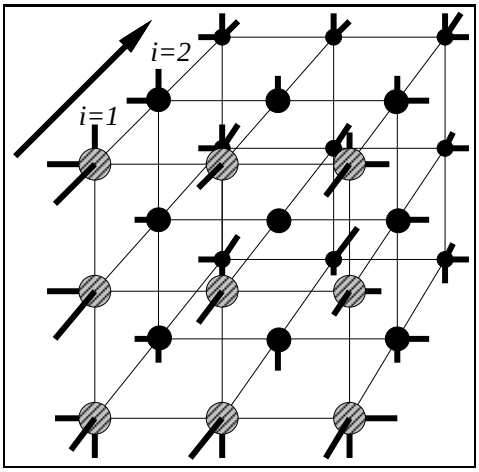


Figure E.2: Example of a N^d switch cluster with $N = 3$ and $d = 3$. Computing nodes are not drawn. The computing nodes are connected on the thick lines from the "surface" switches.

N^d switch clusters for 216 computing nodes.											
d	N	i	N_t	N_r	N_S	Number of computers	Number of switches	Traffic density in λ	Packet-bandwidth	load in $\lambda\overline{x}$	Competition rate
2	*54*	*1*	*56*	*160*	*54*	*216*	*2916*	*0.77*	$4\overline{x}$	*3.08*	*1.04*
2	54	27	108	108	54	216	2916	1	$4\overline{x}$	4	2
3	*6*	*1*	*60*	*156*	*36*	*216*	*216*	*1.2*	$6\overline{x}$	*7.2*	*1.67*
3	6	3	108	108	36	216	216	1.5	$6\overline{x}$	9	3
4	3	1.5	108	108	27	216	81	2	$8\overline{x}$	16	4

Table E.1: Some examples of N^d switch clusters connecting 216 computing nodes.

into link directions. The reference packet bandwidth $\overline{x}$ is the time needed to pass an average packet with the total switch bandwidth. Since this implies a switch with only one link this switch is strictly theoretical. Traffic density and load are found by calculating the data passing a cross section of the switch cluster. The position of the cross section is given by i. In figure E.2 and E.3 we have a cross section at $i = 1$. The traffic we are calculating is the traffic passing first a grey switch and then a black switch. Traffic kept inside the grey zone or kept inside the black zone is not included.

E.2.1 N^d switch clusters connecting 216 computing nodes.

Table E.1 gives the numbers for a two-, a three- and a four- dimensional switch cluster. Values for the links passing the middle cross-section of each cluster are given. These are the links with the highest load. For the two- and three- dimensional clusters values for the links through cross-sections close to one of the edges are also given (written in *italic*) in the table.

In table E.1 N_t is the number of transmitting computers while N_r is the number of receiving computers for the counted communication connections. N_S is the number of links (or more

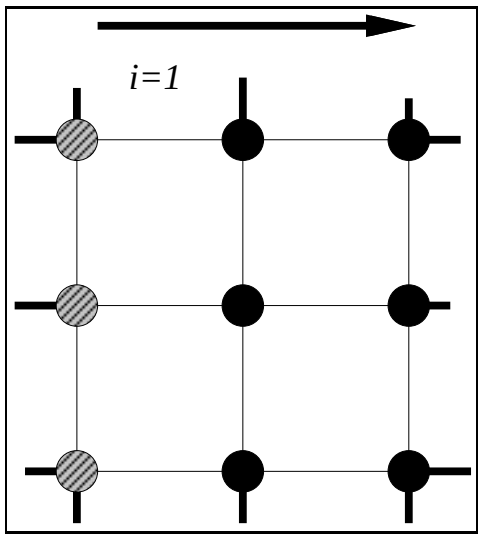


Figure E.3: Example of a N^d switch cluster with $N = 3$ and $d = 2$. Computing nodes are to be connected on the thick lines at the switch cluster edges.

exactly: switches with a transmitting link) through the cross-section. For these topologies the number of switches required is 2916 for a two-dimensional network, 216 for a three-dimensional network and 81 for a four- dimensional network. When designers try to spread the traffic to all links through a cross-section, the traffic density is $d/2$ through the middle links and slightly less through the links close to the edges. The total switch bandwidth has to be divided into $2d$ link directions, resulting in a time of $2d\overline{x}$ to transmit a packet. The load in a link direction out from a switch is $d^2\lambda\overline{x}$ for the middle links. The collection rate, i.e. the average number of hosts communicating through one of the intersection links is d.

For low dimension networks the bandwidth in each link direction is high, resulting in short packet-bandwidth time. Reduction of packet-bandwidth time without increased packet rate gives reduced link load, which shortens the queueing time. Thus less waiting time has to be added to the latency. Also the small number of sources addressing each link reduces the queuing time. Decreasing the number of competitors gives shorter queuing time also for fixed total load since queuing takes place outside the system.

The conclusion is that a low-dimensional network consisting of switches with high bandwidth in each direction has a better performance than a high-dimensional network with lower bandwidth. The only exception is when the queuing time and the packet length-bandwidth relation are so small that the propagation time is dominating. Then the number of switches to be passed should be kept small and a large-dimensional network may be preferred. The high performance of the low-dimensional systems have an important cost: the high number of switches required.

E.2.2 N^d switch clusters for 500, 1000 and 4000 computing nodes.

Table E.2, E.3 and E.4 gives the numbers for switch clusters serving 500, 1000 and 4000 computing nodes respectively. The conclusions are as for the 216 computing node networks. Notice the pay in high number of switches for the advantages of the low dimensional networks.

N^d switch clusters for approximately 500 computing nodes.											
d	N	i	N_t	N_r	N_S	Number of computers	Number of switches	Traffic density in λ	Channel directions	load in $\lambda\overline{x}$	Competition rate
2	*125*	*1*	*127*	*373*	*125*	*500*	*15625*	*0.76*	*4*	*3.04*	*1.02*
2	125	62.5	250	250	125	500	15625	1	4	4	2
3	*9*	*1*	*117*	*369*	*81*	*486*	*729*	*1.1*	*6*	*6.6*	*1.4*
3	9	4.5	243	243	81	486	729	1.5	6	9	3
4	4	2	256	256	64	512	256	2	8	16	4

Table E.2: Some examples of N^d switch clusters for 500 computing nodes.

N^d switch clusters for approximately 1000 computing nodes.											
d	N	i	N_t	N_r	N_S	Number of computers	Number of switches	Traffic density in λ	Channel directions	load in $\lambda\overline{x}$	Competition rate
2	250	125	500	500	250	1000	62500	1	4	4	2
3	13	6.5	507	507	169	1014	2197	1.5	6	9	3
4	5	2.5	500	500	125	1000	625	2	8	16	4

Table E.3: Some examples of N^d switch clusters for 1000 computing nodes.

N^d switch clusters for approximately 4000 computing nodes.											
d	N	i	N_t	N_r	N_S	Number of computers	Number of switches	Traffic density in λ	Channel directions	load in $\lambda\overline{x}$	Competion rate
2	1024	512	2048	2048	1024	4096	1048576	1	4	4	2
3	26	13	2028	2028	676	4056	17576	1.5	6	9	3
4	8	4	2048	2048	512	4096	4096	2	8	16	4

Table E.4: Some examples of N^d switch clusters for 4000 computing nodes.

E.2.3 Comments for general structures.

A natural question is whether any conclusions for network topologies in general can be made from the N^d topologies. An obvious general rule is that more links per switch give less bandwidth per direction. Reduced bandwidth increase both packet bandwidth time and queuing time. Fewer dimensions or connection points require more switches. In general networks the number of switches may be made smaller both for high and low d compared with the N^d clusters.

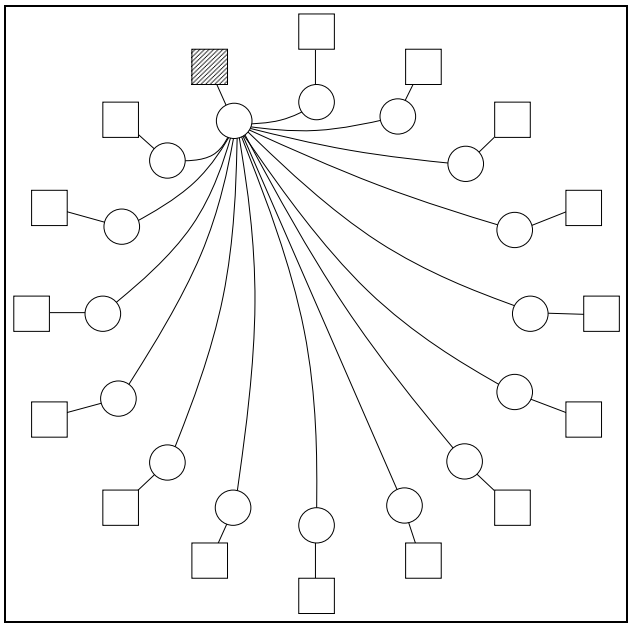


Figure E.4: Private connection network with 16 switches, one for each computing element. Only the connections from the marked host are drawn. The connections from the other hosts are similar.

In the N^d topology the average number of sources addressing a link direction increases with d. This is not a rule for irregular structures. A simple example is a system of 16 switches, each private to one computing element, with private connections to all the other switches. The number of directions is high but only one source is accessing each direction from a switch.

E.2.4 Links and link directions.

In the previous text the term "link directions" has been used frequently. A link direction may contain several parallel links. A switch with 32 physical links may have 8 parallel links in each direction in a two-dimensional network and four parallel links in each direction in a four-dimensional network. Thus the same switch may be used for several dimensions as the topology is reconfigured. A switch may first be used in a multi-dimensional network with links to several net nodes. Then as more switches are added the bandwidth in each direction may be increased by reducing dimensions and merging physical links together.

Appendix F:
State diagram for Input Port without pipelining.

Figure 10.3 shows how data are read out from the Input Port when all data manipulation takes place at the FIFO output. Table F.1 shows the state diagram. The state diagram has 6 states and 32 transitions. Implemented like this the complexity of the state diagram required a clock period of almost 20ns. To make it possible to operate with a faster clock rate the packet data manipulation was pipelined.

A transition can either result in that the state machine enters a new state or that it continues to stay in the present state. The transitions has been numbered. These numbers are given in column 1. The input values which identify a transition are given in columns 3. to 12. They give the present state of the state machines and some single input values. The state after the transition is given i column 14.

The **F**-register is the 9-bit word on the output of the Input Port FIFO. The **S**-register contains the packet word following the word in the **F**-register. The **N** buffer contains the five next bits to be forwarded to the CSU on the following negative clock edge. The **P** buffer contains the five next bits to be forwarded to the CSU on the following positive clock edge.

The columns have the following meaning:

1. Transition number.
2. (Referance to layout).

3.-12. Input signals. All signals are active high with the following meanings:

3. ig/Incoming guard: The CSU is ready for receiving (more) data.
4. Timeout/Reset/Fatal error: The connection will be broken and the state machine enters the reset state.

5.-8. Concerns the contents of the **S**-register.

5. SValid: The **S**-register contains a valid symbol.
6. SData: The **S**-register contains a valid data symbol i.e. the **S** register contains a valid symbol and this symbol is a data symbol.
7. Sjc=0: The **S**-register contains a valid symbol and this symbol is a data symbol and the value of the upper half is zero.
8. Sdf: The **S**-register contains a *df*-symbol.

9.-11. Concerns the contents of the **F**-register.

9. FValid: The **F**-register contains a valid symbol.
10. Fbop: The **F**-register contains a *bop*-symbol.
11. Feop: The **F**-register contains a *eop*-symbol.
12. The present state: The present state of the state machine.
13. (Reference to layout not used in this thesis).

14.-19. Output signals.

14. The state after transition.
15. -1: The upper part of the **F**-register load the **S**-register through the subtract-by-one logic. This is used to subtract the **jump counter** by one.
16. ShiftS: Shift new value into the **S**-register. This happens either when the **S**-register is empty or when the contents of the **S**-register shall be forwarded to the **F**-register.
17. ShiftF: Shift a new value into the **F**-register from the **S**-register.
18. Timeout run: This indicates that the timeout counter shall count. If this counter reaches the timeout limit this will result in a timeout. A low "timeout run" signal will reset the timeout counter to zero.
19. Bus instruction: The output buffers **P** and **N** can be loaded by fixed values and the upper and lower parts of the **F** and **S** registers in seven different ways. This column gives which of the seven possibilities this transition use.
20. This column gives a short description of the transition. The meaning of $v(x)$ is that the state machine will stay in the present state while waiting for x. The state machine may wait for a ready (ig) signal from the CSU or it may wait because of an empty register (F or S). $-y-$ indicates that y is forwarded to the CSU.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
		ig	Time-out alarm	S-reg SValid Sdata Sjc=0 Sdf				F-reg Fvalid Fbop Feop			Present state		Next state	-1	Shift in S	 F	Time-out run	bus instruc.	

Name of present state: No connection

1	1		1								0.	0	4.	0	0	0	0	eop	Timeout
2	1		0	1	1	0		1	1		0.	1	a1.	0	0	0	0	A4	-a4-
3	2		0	1	1	1		1	1		0.	2	0.	0	1	1	0	Idle	dism.
4	3		0	0	0			1	1		0.	3	0.	0	1	0	1	Idle	v(S)
5	4		0						0		0.	4	0.	0	1	1	0	Idle	dism.
6			0	1	0			1	1		0.	5	0.	0	1	1	0	eop	Error

Name of present state: Waiting for channel confirmation

7	1		1	< data >				<*bop*>			a1.	0	4.	0	0	0	0	eop	Timeout
8	2	1	0	< data >				<*bop*>			a1.	1	a2.1.	1	1	1	0	ord-1	-bop-
9	3	0	0	< data >				<*bop*>			a1.	2	a1.	0	0	0	1	Idle	v(ig)

Name of present state: Transmit jc

10	1		1					< data >			a2.1.	0	4.	0	0	0	0	eop	Timeout
11	2	0	0	1				< data >			a2.1.	1	a2.1.	0	0	0	1	Idle	v(ig)
12	2	0	0	0				< data >			a2.1.	2	a2.1.	0	1	0	1	Idle	v(ig)
13	3	1	0	0				< data >			a2.1.	3	a2.1.	0	1	0	1	Idle	v(S)
14	4	1	0	1	1			< data >			a2.1.	4	a2.2.	0	1	1	0	jc	-jc-
15		1	0	1	0			< data >			a2.1.	5	4.	0	0	0	0	eop	Error

Name of present state: Transmit address

16	1		1					< data >			a2.2.	0	4.	0	0	0	0	eop	Timeout
17	2	1	0	0				< data >			a2.2.	1	a2.2.	0	1	0	1	Idle	v(S)
18	3	1	0	1			1	< data >			a2.2.	2	3.	0	1	1	0	ret	-ret-
19	4	1	0	1	1			< data >			a2.2.	3	a2.2.	0	1	1	0	adr	-adr-
20	5	0	0	1				< data >			a2.2.	4	a2.2.	0	0	0	1	Idle	v(ig)
21	5	0	0	0				< data >			a2.2.	5	a2.2.	0	1	0	1	Idle	v(ig)
22		1	0	1	0		0	< data >			a2.2.	6	4.	0	0	0	0	eop	Error

Name of present state: Transmit data

23	1		1								3.	0	4.	0	0	0	0	eop	Timeout
24	2	1	0							1	3.	1	4.	0	1	1	0	eop	-eop-
25	3	0	0					0			3.	2	3.	0	1	1	1	Idle	v(ig)
26	3	0	0	0				1			3.	3	3.	0	1	0	1	Idle	v(ig)
27	3	0	0	1				1			3.	4	3.	0	0	0	1	Idle	v(ig)
28	4	1	0					1		0	3.	5	3.	0	1	1	0	ord	-data-
29	5	1	0					0		0	3.	6	3.	0	1	1	1	Idle	v(F)

Name of present state: Disconnect connection

30			1								4.	0	4.	0	0	0	0	eop	Timeout
31	1	1	0								4.	1	4.	0	0	0	0	eop	v(not ig)
32	2	0	0								4.	2	0.	0	0	0	0	Idle	idle

Table F.1: State table for Input Port supporting single input — single output connections.

Some of the values in the state diagram are implicit: e.g. the 'FValid'-signal will always be high when the 'Fbop'-signal is high. If the 'Fbop'-signal is used to identify a transition the implicit 'FValid' is marked by a small *1*.

When we enter the **Waiting for channel confirmation** state we know that the **S**-register will contain a data symbol and that the **F**-register will contain a *bop*-symbol. This will continue to be true as long as we stay in this state. This implicit information is shown in the state diagram in the column below the **S**-register and the **F**-register. The same knowledge of the **F**-registers contents is shown for the **Transmit jc** and **Transmit address** states.

The timeout/reset signal works as a collected reset signal for a number of error causes. When this signal goes high this will independently of the present state result in that the next state is **Disconnect connection**. Causes that may result in a reset signal are:
- Initial reset,
- Timeout counter reached timeout limit,
- Wrong control symbol inside the **switch data field**,
- Erroneous state (depending on how the state machine has been implemented).

Appendix G:
Layout examples.

ipi	tfifo	fifoword	fifocont5
			fifomem5
			fifomem4
		fifowordend	fifocontend
			fifomem5
			fifomem4
	mastbuff		
	word13a	tspc2s	
	seca4	*mastbuff*	
		muxsa4	
		dec	
		muxs	
		passa4	
		muxa41	
		nora4	
		muxfa4o	
		word13r	*tspc2s*
		word13a	*tspc2s*
	secout	mastbuff	
		word12b	*tspc2s*
		rqmux	rq1
			rq0
		qpmux	qp0
			qp1
		word9b	*tspc2s*
		word12mr	tspc2tbop
			tspc2s
		nand33	
		nand22	
		sreg2	
		nand55	
		Master	
	secnm	nmfshift	
		muxnmct	
		word13a	*tspc2s*
		nmmux	
		muxnmr13	muxnmr
		secnmcont	
		tspcr	
op	opoinv10	opoinv	
	opoutmux	opoutmux1	
	opcont		
	fifowordend	*fifocontend*	
		fifomem5	
		fifomem4	
	fifoword	*fifocont5*	
		fifomem5	
		fifomem4	
	opinpmux	tspc2pt	
		tspc2nt	
		tspc2p	
		tspc2n	
		opinpmux1	
		fifoword	*fifocont5*
		fifomem5	
		fifomem4	

Table G.1: Cell hierarchy for the Input and Output Port. Cell names are written with roman font the first time they are mentioned, later they are written in italic.

G.1 The high speed FIFO: tfifo.

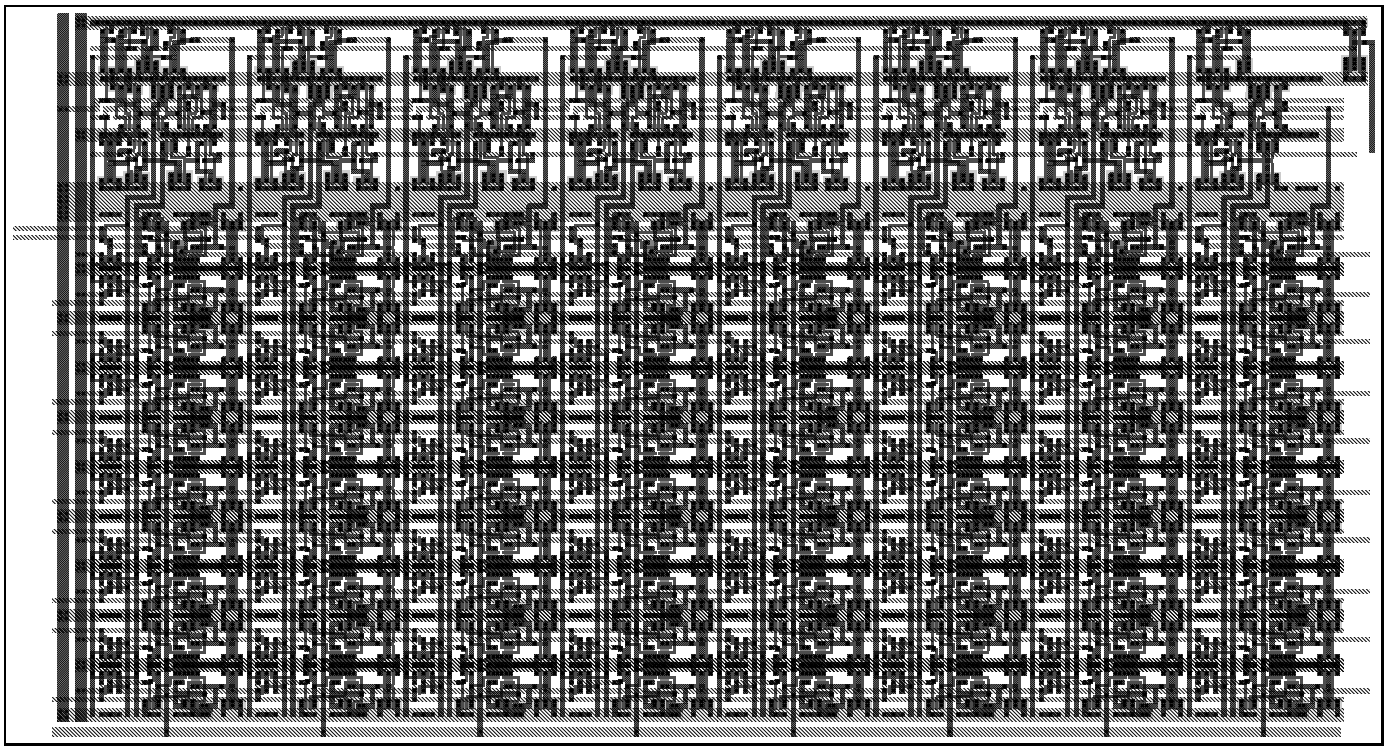


Figure G.1: The layout shows a fall-through high-speed FIFO. Data are read out on the right side. New words arrive from the left and fill the rigthmost positions first. Each word contains 9 bits and a valid bit indicating whether or not the word contains valid data. A control block on top of each word register decides whether the register should copy the bus, copy the next register or refresh. With typical $1.2\mu m$ *parameters the longest signal path is* $3.6ns$*. Each word register contributes with capacitive load on the incoming bus and on the shift and shiftb signals, all going through all word registers. Thus the clock rate will also depend on the delays through the buffers driving these lines. The size of each word register is* $0.047mm^2$*.*

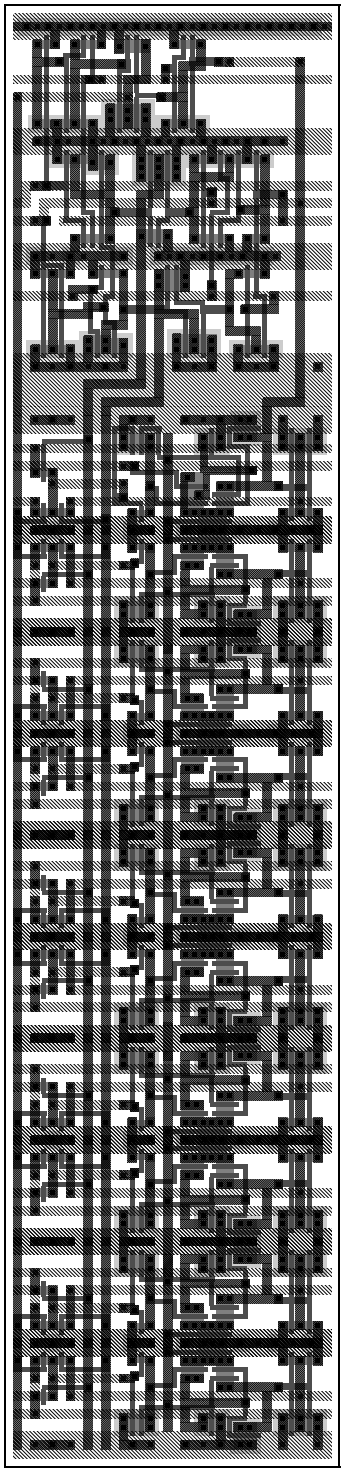


Figure G.2: Word register with control block, valid bit and 9 bits of data.

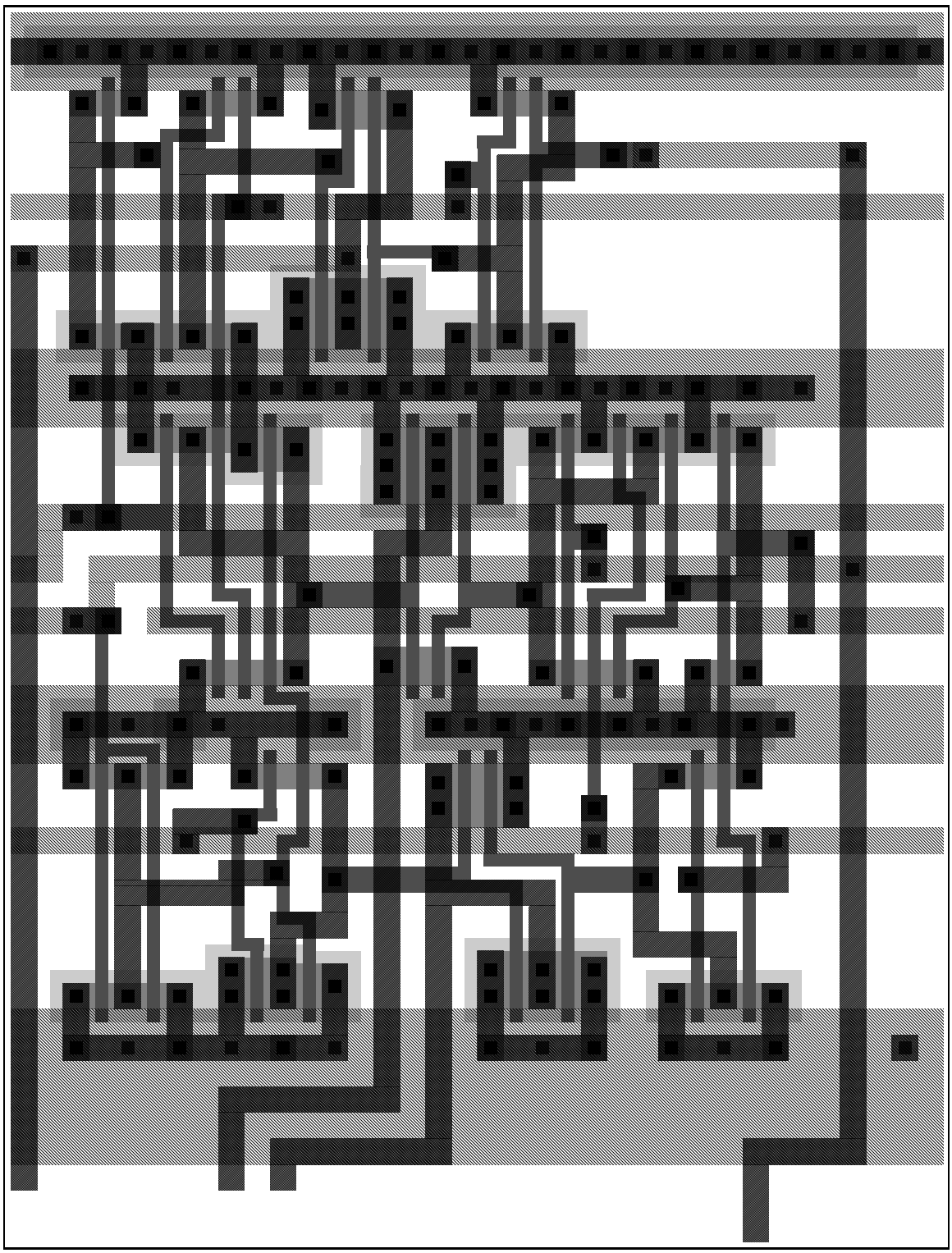


Figure G.3: Control logic for word register. The gates in the upper row generate the Read(i+1) signal for reading of succeeding words. The gates in the middle row generate the signal for reading the incoming bus Readbus. In the bottom row the Read(i) (refresh) signal is generated. The horisontal signal through the upper row is shift while the horisontal signal through the bottom row is shiftb. Boolean equations: Read(i+1) = V(i+1) · shiftb + shift · $\overline{V(i)}$*; Readbus =* $\overline{V(i+1)}$*· shift · V(i) +* $\overline{V(i)}$*· shift · V(i-1);Read(i) = shiftb · V(i) + shiftb ·* $\overline{V(i-1)}$

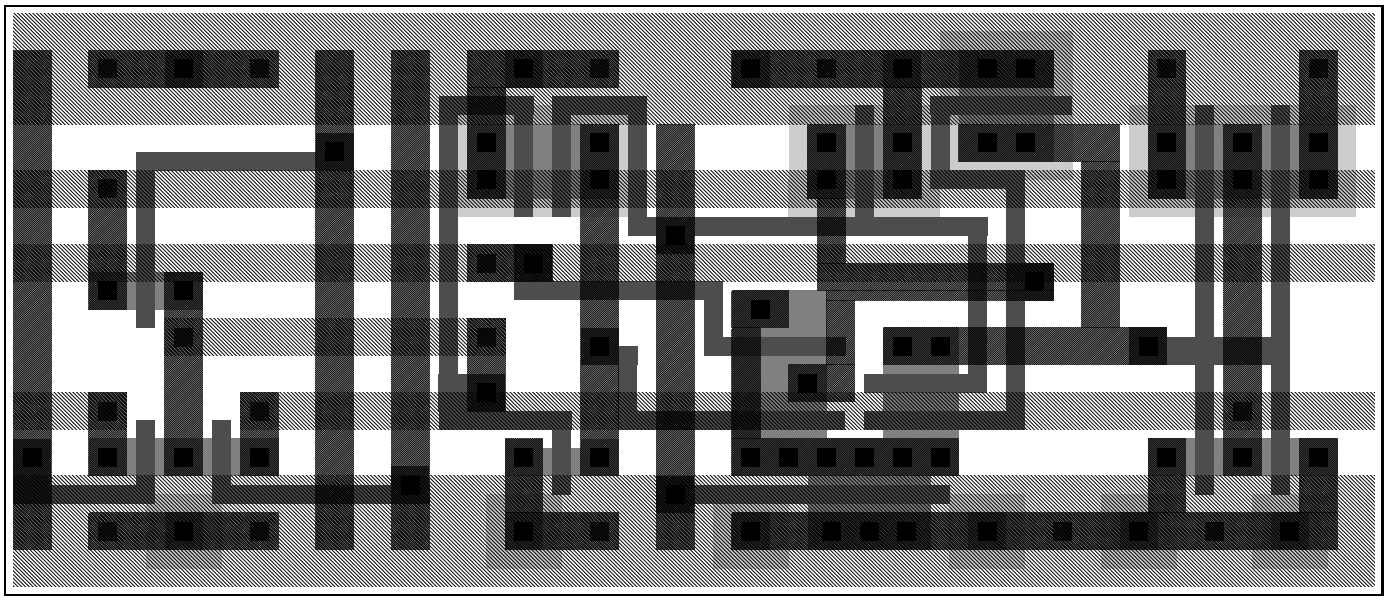


Figure G.4: The valid bit. All latches are dynamic single-phase latches. The second horisontal signal line from the top is used to reset the valid latches initially.

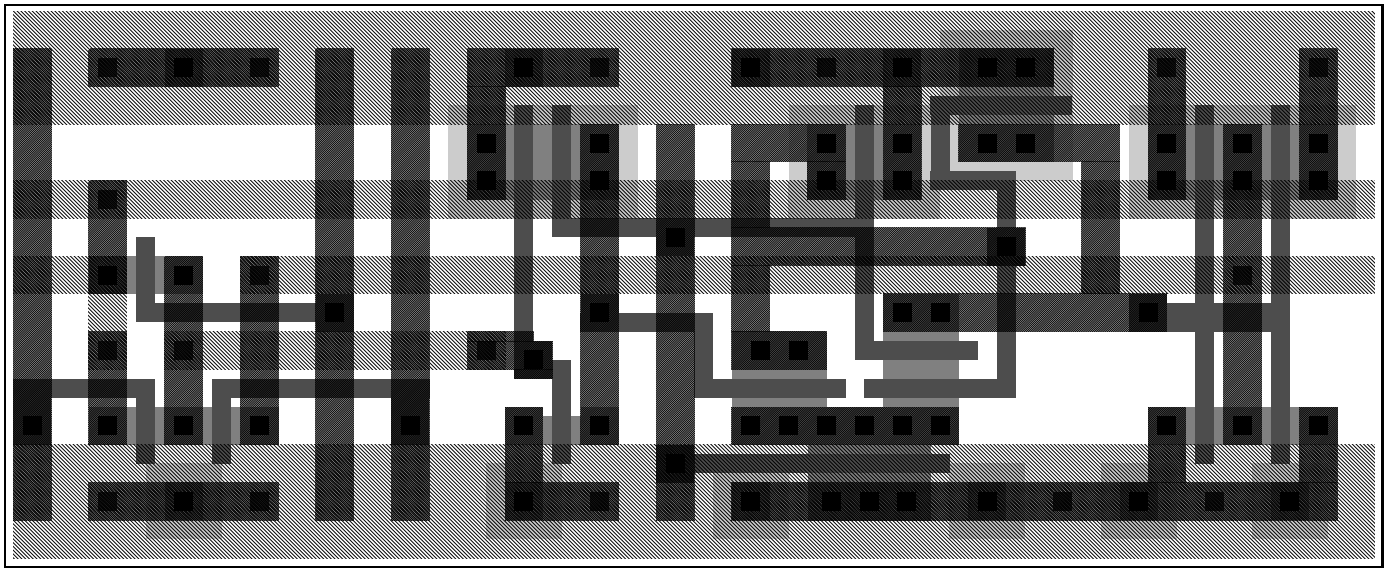


Figure G.5: Layout of each of the remaining 9 bits of each word.

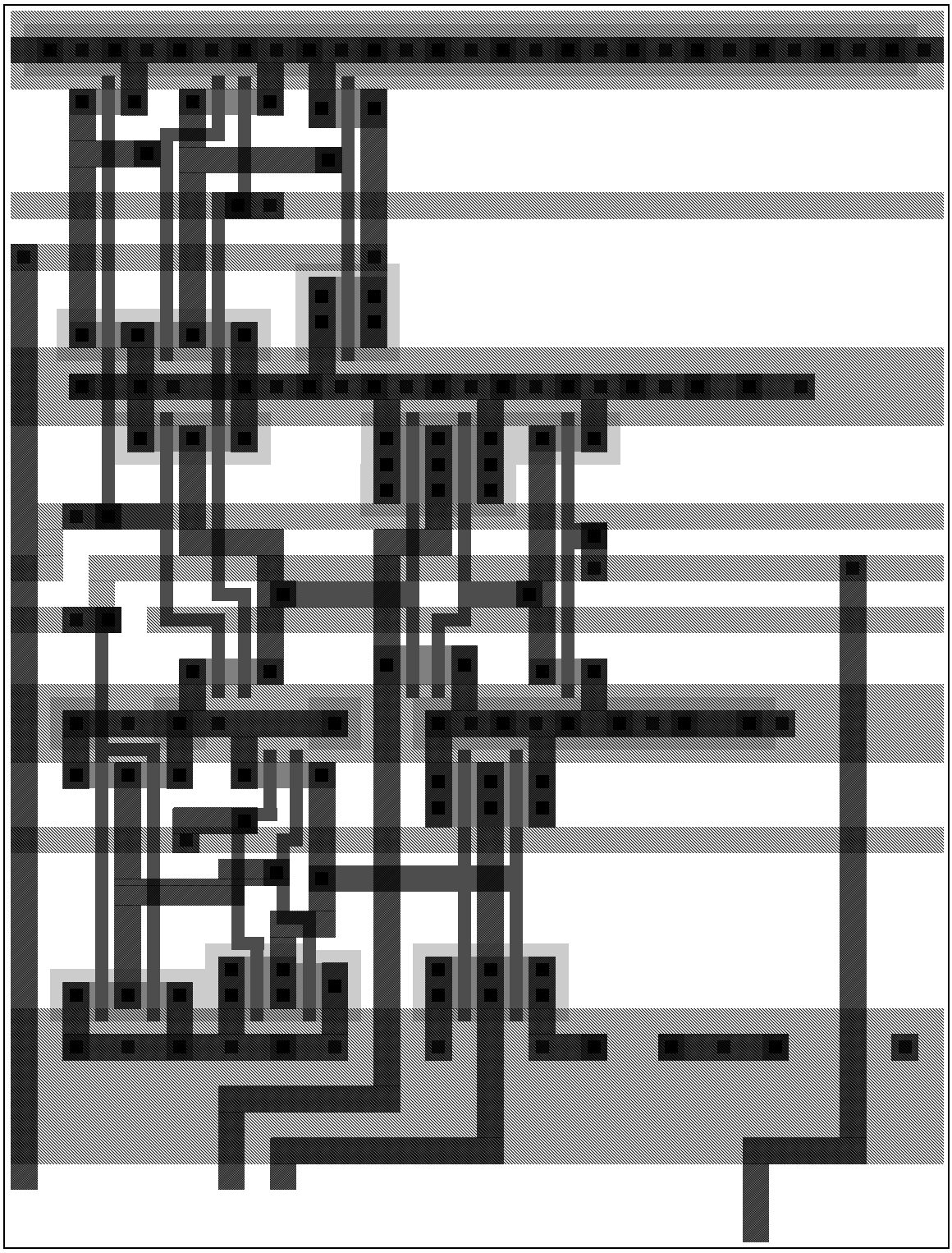


Figure G.6: Control logic for the word closest to the output (word 0). Boolean equations: Read(i+1) = V(i+1) · shift; Readbus = shift · $\overline{V(i+1)}$ + $\overline{V(i)}$; Read(i) = shiftb · V(i)

G.2 RAM.

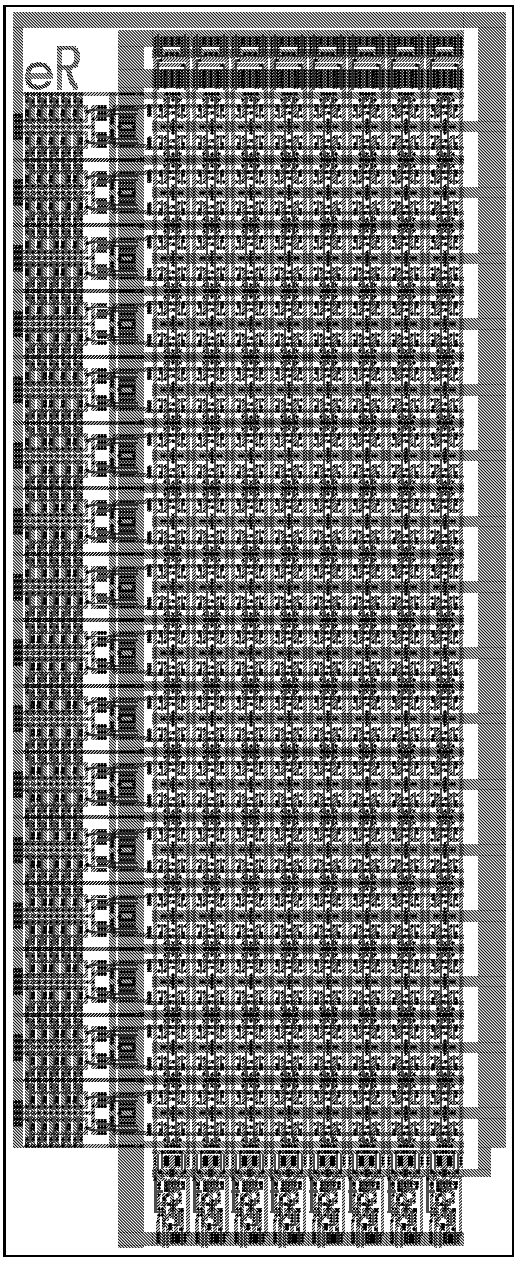


Figure G.7: Single-port RAM with 32 words with a word length of eight bits.

G.3 Output Port.

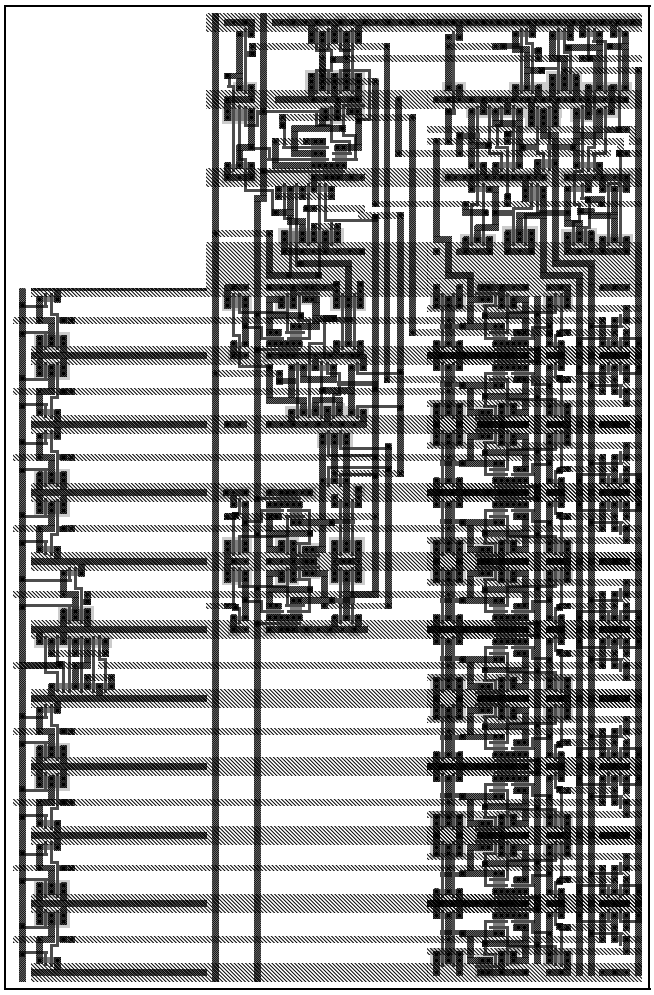


Figure G.8: Output part of the Output Port. The rightmost column is a part of the FIFO. It is important for the FIFO speed that the shift signal is immediately available after a clock edge. To obtain this the "roots" of the shift signal are found two time periods back and calculated during these clock periods. Thus not only the FIFO contents closest to the output are required. The middle block is logic to secure correct operation if the Output Port FIFO should be filled to its limit. The leftmost column is a mux for shifting between the FIFO output and an idle word for flow control symbol insertion. (The missing connections and contacts are drawn at a level higher up in the hierarchy.)

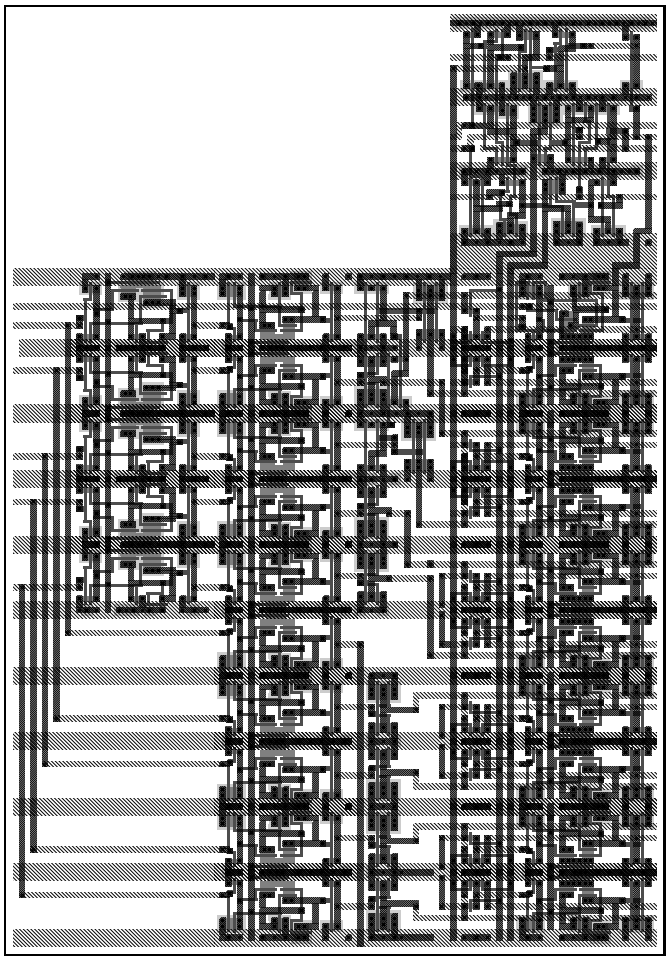


Figure G.9: Input part of Input Port. The leftmost column contains 5 latches clocking on the negative clock edge. The second column has 10 latches clocking on the positive clock edge.

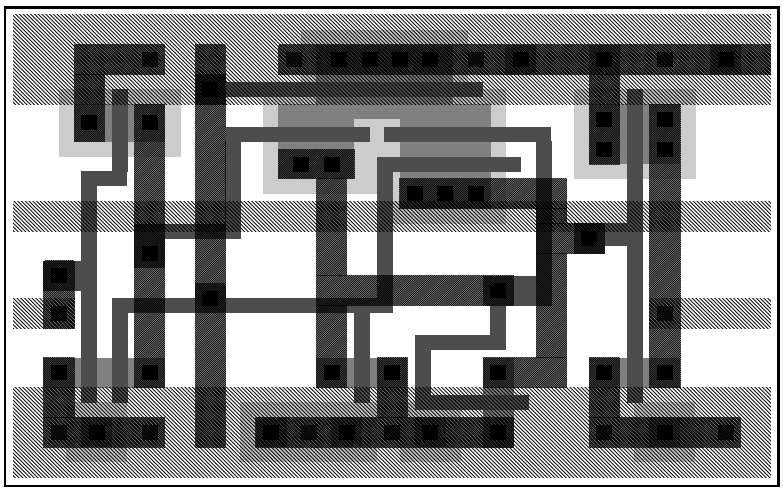


Figure G.10: Latch clocking on the negative clock edge.

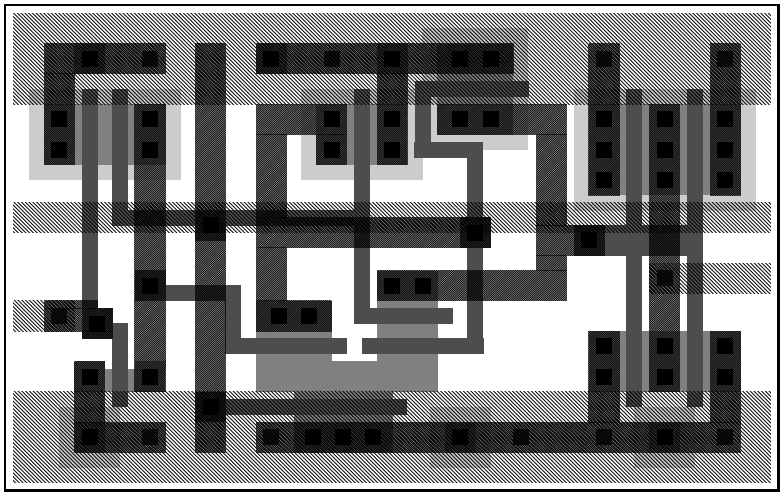


Figure G.11: Latch clocking on the positive clock edge.

G.4 Relative sizes of switch elements.

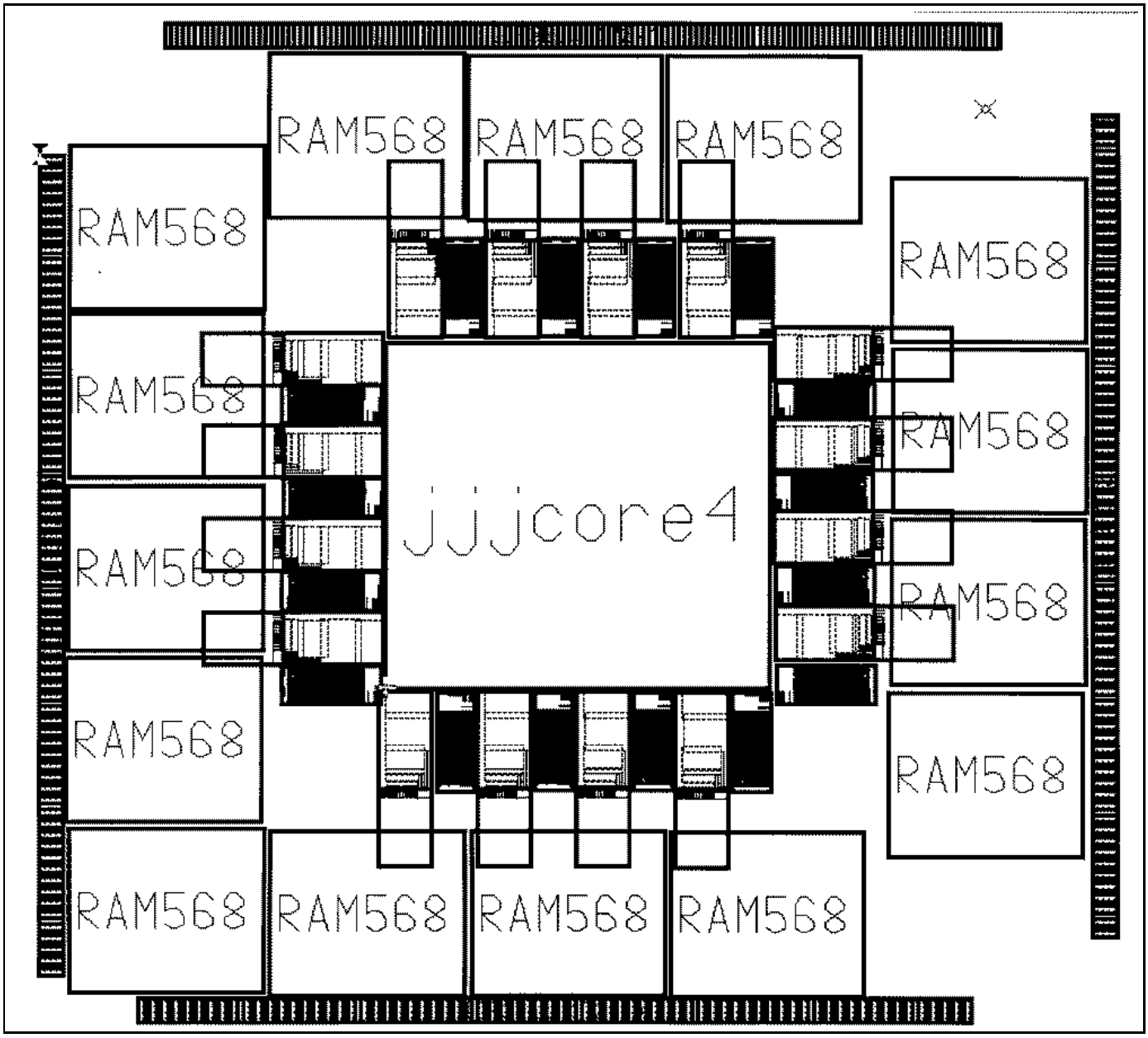


Figure G.12: The figure shows a screen dump from the Mentor IC-station. The jjjcore4-block is an auto-routed version of the CSU. Logical simulations show correct behaviour, but the layout has not been optimised concerning internal routing. Anyway the size, $40mm^2$, should be correct. Outside of the jjjcore-block we have 16 input and output ports as described elsewhere in this thesis. The area of the Input Ports, the Output Ports and the jjjcore4-block is $100mm^2$. Outside of the Input and Output Ports we have 16 one-port RAM blocks of 568 words. The area of the RAM modules and the contents inside is $290mm^2$. Pads make up the outermost frame. The pads have a pitch of $200\mu m$. There are 18 pads for each channel, 9 for the incoming direction and 9 for the outgoing direction. The complete circuit has a size of $18mm \times 17.4mm = 320mm^2$, which is too large for present technologies. There is a potential for reduction of the size of the Input Ports, the Output Ports and the RAM blocks: They were drawn for $1.2\mu m$ in another technology (Orbit) on another design system (Magic). When converted to Mentor the layout had to be scaled from $1.2\mu m$ to $2.0\mu m$ to satisfy all design rules. The simulations presented in this thesis were performed with the "old" technology rules and parameters. Thus they will not be valid in the "new" technology without further optimisation.

Appendix H:
Flow-control based input port buffer size.

There are several factors which influence on the preferred size of the SWIPP input buffers. One such factor is the flow control reaction time. This appendix discusses minimum requirements on the buffer size based on the flow control reaction time when the transmitting and receiving net nodes have different clock rates. In SWIPP probably it will be most common to let all channels have the same bandwidth. In some cases it may be advantageous to give some parts of the network higher bandwidth due to larger traffic. It may also be preferred to let some parts have lower bandwidth to save power.

The flow control reaction time is important for the sizing of the SWIPP Input Port buffers. In most systems a positive flow control signal will allow a specific amount of data (for example 8 bytes) to be transmitted from the upstream net node. In SWIPP a transmitting buffer can forward data until a negative flow control signal is received. Thus a new flow control signal is not required before the receiving buffer capacity is almost exceeded. The flow control reaction time will decide when the negative flow control signal has to be transmitted. The flow control reaction time is depending both on the transmitting clock frequency and on the receiving clock frequency. When different net nodes are allowed to have different clock frequencies the flow control reaction time will be variable between the different pairs of net nodes. In addition to different clock frequencies, different physical distances between net nodes contributes to the variation of flow control reaction times.

H0.1 An equation for the flow control reaction time.

We will in the following give an estimate of the flow control reaction time.

The structure of the tables and equations on the following pages are copied from the thesis of one of the master students supervised by this author ([86]). New values have been inserted. These values are partially based on exact values and partially on the designer's own estimates.

Figure H.1 shows the different delays of the flow control reaction time. These delays will be defined in the following.

The asynchronous FIFO as it is implemented by Røine contains two counters, one at the input side and one at the output side of the main buffer element. The difference between these two counters decides whether the FIFO is more or less than "almost full". This information is needed to generate the flow control signal.

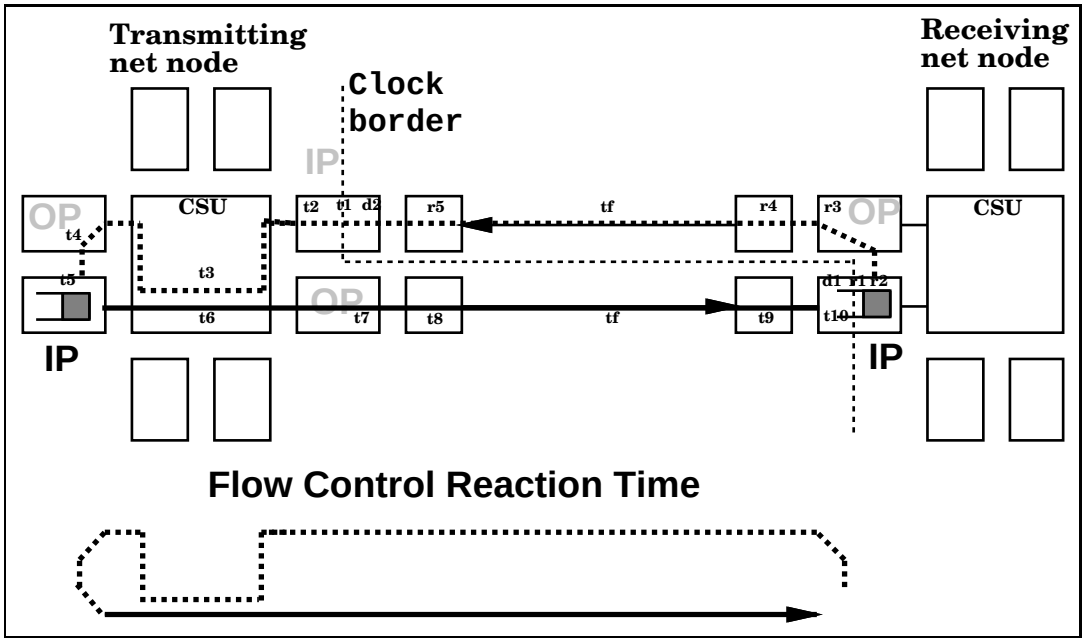


Figure H.1: The different delays constituting the flow control reaction time.

Definition of time delays: d[1:2], r[1:5], tf and t[1:9].

d1: is the delay from a symbol arrives at the Input Port until the counter at the input side of the FIFO is updated according to this symbol. This delay has been simulated to $d_1 \approx 40ns$ ([86] p.17).

r1 = 6: The input counter has to be transferred over the clock border to the output side of the buffer. This may take up to 6 receiver clock periods.

r2 = 2: Two clock periods are used to compare the two counters and generate flow control signals according to the difference between them.

r3 = 2: Two clock periods are used to transfer the flow control signal through the Output Port.

r4 = 2: The generation of fibre symbols for the optical fibre takes two clock periods.

tf: The time a symbol needs to propagate along the optical fibre is denoted by **tf**. **tf** can be expressed by $\frac{n}{c}l$ where n is the refractive index of the fibre, c is the speed of light in vacuum and l is the length of the fibre. $\frac{c}{n}$ is the speed of light inside the fibre.

r5 = 2: In the other end of the fibre two new clock periods are needed to propagate the signal through the Optical Module.

All of the steps **r1, r2, r3, r4** and **r5** are clocked by the clock of the receiver. We name the total steps clocked by the receiver clock N_r and get N_5 =**r1 + r2 + r3 + r4 + r5**. For the values given above we will have $N_r = 14$.

d2: A time delay of $30ns$ is needed to propagate the flow control signal through the input part of the elastic FIFO.

t1 = 6: Up to 6 clock periods of the transmitter clock rate are needed to transfer the flow control signal from the clock of the receiver to the clock of the transmitter.

t2 = 1: An additional clock period is used to propagate the clock signal through the Input Port.

t3 = 1: One clock period is needed to transfer the flow control signal through the CSU.

t4 = 1: The flow control signal has to be transmitted via the Output Port. This takes one clock period.

t5 = 2: The transmitting Input Port needs two clock periods before the packet stream is changed according to the flow control signal.

In the following we will follow the last transmitted data symbol not influenced by the arriving flow control signal.

t6 = 1: The last data symbol needs one clock period to propagate through the CSU.

t7 = 2: Two more clock periods are needed to forward the data symbol through the Output Port.

t8 = 2: The Optical Module requires two clock periods for generation of symbols for the optical fibre.

tf: The propagation time for the data symbol down the fibre is equal to the propagation time for the flow control signal in the opposite direction.

t9 = 2: The last two clock periods are used to transfer the data symbol through the Optical Module.

All of the steps **t1-t9** are clocked by the clock of the transmitter. We name the number of clock periods clocked by the transmitter N_t and have $N_t = 18$.

From the steps above we can define the **flow control reaction time** t_{fcrt} in the following equation:

$$t_{fcrt} = \frac{N_t}{f_t} + d_1 + d_2 + \frac{N_r}{f_r} + 2 \cdot t_f \tag{H.16}$$

f_t is the clock frequency of the transmitting net node while f_r is the clock frequency of the receiving net node. As stated above $N_t = 18$, $N_r = 14$, $d_1 = 40ns$ and $d_2 = 30ns$. t_f is the delay due to the propagation time along the fibre.

H1 The size of the Input Port buffer.

In the following we will find equations for absolute and recommended minimum buffer sizes for the SWIPP flow control strategy.

Figure H.2 gives some parameters for the Input Port buffer. At the top of the buffer we find the "almost full" mark.

On the bottom of figure H.2 we find N_{min}, N_{exit} and N_{good} which will be discussed in the following.

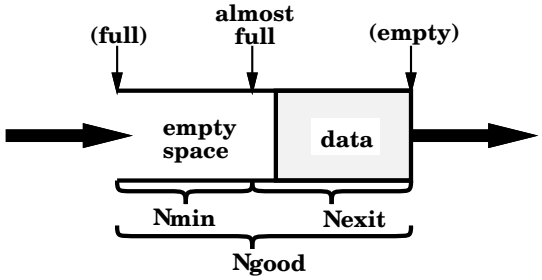


Figure H.2: Some parameters describing the Input Port buffer.

H1.1 N_{min}: The minimum buffer size.

When the "almost full" mark is reached, the remaining part of the buffer has to be large enough to store all data received until the packet stream is stopped. The capacity of this part of the buffer is equal to the number of symbols that the transmitting net node can forward during one flow control reaction time. This capacity is given by the following equation:

$$N_{min} = t_{fcrt} \cdot f_t = N_t + d_1 \cdot f_t + d_2 \cdot f_t + N_r \cdot \frac{f_t}{f_r} + 2 \cdot t_f \cdot f_t \qquad \text{(H.17)}$$

The table below shows the size of N_{min} for some pairs of transmitter clock rate and receiver clock rate. We can see from the table that N_{min} is large when the transmitter frequency is much larger than the receiver frequency.

N_{min} in number of symbols.							
Receiver frequency, f_r	Transmitter frequency, f_t, in millions of symbols						
in millions of symbols	1	2	5	10	20	50	100
1	32.07	46.14	88.35	158.7	299.4	721.5	1425
2	25.07	32.14	53.35	88.7	159.4	371.5	725
5	20.87	23.74	32.35	46.7	75.4	161.5	305
10	19.47	20.94	25.35	32.7	47.4	91.5	165
20	18.77	19.54	21.85	25.7	33.4	56.5	95
50	18.35	18.7	19.75	21.5	25.0	35.5	53
100	18.21	18.42	19.05	20.1	22.2	28.5	39
Add pr. meter of fibre	0.010	0.021	0.052	0.103	0.207	0.517	1.033

H1.2 N_{exit}: When transmission restarts.

When an occupied output channel gets ready data can be read out of the FIFO. When the "almost full" mark is passed in the decreasing direction a "start transmission" flow control signal will be transmitted upstream.

To assure high utilisation of the reserved output channel the network system should be capable of supporting the output channel with packet symbols until the packet has reached its end. To make this possible the part of the buffer within the limit of the "almost full" mark must be able to hold enough packet data to keep the output channel busy until more data arrive from the next buffer. This buffer capacity is equal to the number of symbols that the receiving net node can

forward during one flow control reaction time. This capacity is given by the following equation:

$$N_{exit} = t_{fcrt} \cdot f_r = N_s \cdot \frac{f_r}{f_t} + d_1 \cdot f_r + d_2 \cdot f_r + N_r + 2 \cdot t_f \cdot f_r \qquad \text{(H.18)}$$

The table below shows some values of N_{exit} for some pairs of clock rates for the transmitting and receiving net nodes. We see from the table that N_{exit} is large when the receiver frequency is much larger than the transmitter frequency.

N_{exit} in number of symbols.								
Receiver frequency, f_r	Transmitter frequency, f_t, in millions of symbols							Add pr. meter
in millions of symbols	1	2	5	10	20	50	100	of fibre
1	32.07	23.07	17.67	15.87	14.97	14.43	14.25	0.010
2	50.14	32.14	21.34	17.74	15.94	14.86	14.5	0.020
5	104.35	59.35	32.35	23.35	18.85	16.15	15.25	0.052
10	194.7	104.7	50.7	32.7	23.7	18.3	16.5	0.103
20	375.4	195.4	87.4	51.4	33.4	22.6	19	0.207
50	917.5	467.5	197.5	107.5	62.5	35.5	26.5	0.517
100	1821	921	381	201	111	57	39	1.033

H1.3 N_{good}: A good buffer size.

A good buffer size would be a size equal to or larger than $N_{min} + N_{exit}$.

$$N_{good} = N_{min} + N_{exit} = N_t + N_r + (d_1 + d_2)(f_r + f_t) + N_t \frac{f_r}{f_t} + N_r \frac{f_t}{f_r} + 2t_f(f_r + f_t) \qquad \text{(H.19)}$$

The table below shows some clock rate pairs for the transmitter and receivers net node.

N_{good} in number of symbols.								
Receiver frequency, f_r	Transmitter frequency, f_t, in millions of symbols							Add pr. meter
in millions of symbols	1	2	5	10	20	50	100	of fibre
1	64.14	69.21	106.02	174.57	314.37	735.93	1439.25	0.010
2	75.21	64.28	74.69	106.44	175.34	386.36	739.5	0.021
5	125.22	83.09	64.7	70.05	94.25	177.65	320.25	0.052
10	214.17	125.64	76.05	65.4	71.1	109.8	181.5	0.103
20	394.17	214.94	109.25	77.1	66.8	79.1	114	0.207
50	935.85	486.2	217.25	129	87.5	71	79.5	0.517
100	1839.21	939.42	400.05	221.1	133.2	85.5	78	1.033
Add pr. meter of fibre	0.010	0.021	0.052	0.103	0.207	0.517	1.033	

The buffer requirements increase fast with differences in clock rate. To reduce the demand for large buffers we may decide that the difference in clock rate between two neighbour net nodes is not allowed to be larger than 10 times. The area where the difference in clock rates is less than 10 times is limited by two staircase lines in the table.

The right column and bottom row show how much has to be added for each meter of fibre length. (Both values have to be added.)

We see from the previous table that with equal transmitter and receiver frequency a buffer with a capacity of 80 symbols would be sufficient. If the difference in transmitter and receiver

frequencies is allowed to increase up to 10 times, the buffer space must be increased to 230 symbols. These numbers come from evaluation of the **flow control reaction time** only. Network traffic behaviour should also be considered. The buffer size must be increased with a number of symbols depending on the fiber length. For long fibres at high clock rate the addition is highly significant. A fiber with a length of 200 m increases the "good" buffer size with 414 symbols at a transmitter and receiver clock rate of 100 million symbols per second.

H1.4 A better buffer size ?

If packets are larger than the buffer capacity, parts of a packet that is stopped will be stored in more than one net node. Thus a line of blocked packets may influence more switches. An increase of the buffer sizes will make the contention problem more local.

We may find from the traffic pattern that buffers have to be large enough to store a number of packets. On the other hand we may also conclude from the traffic pattern and applications that it is acceptable that more channel segments are influenced. An indication of an initial buffer size may be found by simulating the network with the expected traffic pattern. Later experience will often indicate that other buffer and/or packet lengths should have been chosen. This is both because it is difficult to predict the traffic pattern of an application in advance, and that in most cases the applications performed on a network are changing.

H2 An alternative solution with clock borders in OP.

This section describes a solution where a clock border is moved from the Input Port against the packet stream to the Output Port in the transmitting end of the fibre. This solution is not as sensitive with regard to differences in clock frequencies as the solution described in the previous part of this chapter. Thus smaller buffers can be used when the transmitter and receiver clock frequencies are different.. For some implementations it is an advantage that no part of the receiving net node uses the clock of the transmitting net node. This may be the case when the transmitting net node has a high clock frequency and we want the receiving net node to have a lower clock frequency (e.g. to attain a low power consumption).

Figure H.3 illustrates the principle of moved clock border. The synchronous buffer of an Output Port has been increased and changed into an elastic buffer. This buffer forwards packet symbols according to received flow control signals. It also transmits flow control signals upstream depending on its own state. The receiving Input Port has the same clock for the input and output side. Hence the Input Port buffer could be exchanged with an ordinary synchronous buffer.

H2.1 Flow Control Reaction Time # 1.

Figure H.3 shows the delay elements of Flow Control Reaction Time # 1. The elements are similar to what has been described before and will not be further explained.

$$t_{fcrt1} = N_{fcrt1}/f_r + d_1 + 2t_f \tag{H.20}$$

N_{min} and N_{exit} for **IP R** both have the same size. This size is given by the following equation.

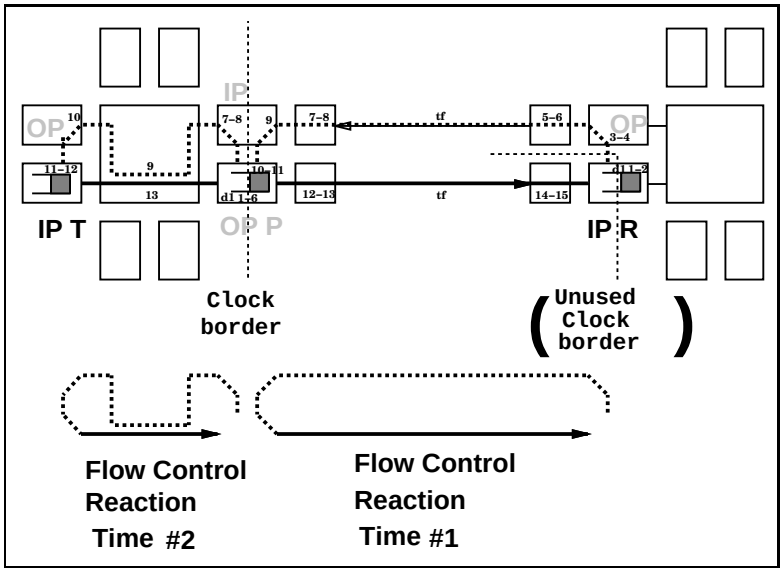


Figure H.3: Switch with clock border through Output Port

$$t_{fcrt1} f_r = N_{fcrt1} + d_1 f_r + 2 t_f f_r \tag{H.21}$$

t_{fcrt1} is the flow control reaction time for circle 1. f_r is the clock rate of the receiving node. N_{fcrt1} is the number of clock steps needed around the flow control reaction circle number 1. N_{fcrt1} is equal to 15. d_1 is $30ns$ as described for the ordinary implementation. As before, t_f is the propagation time along the fibre.

IP R N_{min} = IP R N_{exit}							
	Receiver frequency, f_r, in millions of symbols						
	1	2	5	10	20	50	100
$N_{min} = N_{exit}$	15.03	15.06	15.15	15.3	15.6	16.5	18.0
N_{good}	30.06	30.12	30.30	30.6	31.2	33.0	36.0
Add pr. meter of fibre	0.010	0.021	0.052	0.103	0.207	0.517	1.033

H2.2 Flow Control Reaction Time # 2.

The **flow control reaction time** for circle 2 is given by the following equation.

$$t_{fcrt2} = N_{fcrt2}/f_t + d_1 \tag{H.22}$$

The size of the OP P N_{exit} is given by the following equation.

$$t_{fcrt2} f_t = N_{fcrt2} + d_1 f_t \tag{H.23}$$

f_t is the clock frequency of the transmitting net node. $N_{fcrt2} = 13$ and $d_1 = 30ns$.

IP P N_{min} = IP P N_{exit}							
	Receiver frequency, f_r, in millions of symbols						
	1	2	5	10	20	50	100
$N_{min} = N_{exit}$	13.03	13.06	13.15	13.3	13.6	14.5	16.0
N_{good}	26.06	26.12	26.30	26.6	26.2	28.0	32.0
Add pr. meter of fibre	0.010	0.021	0.052	0.103	0.207	0.517	1.033

H3 Concluding remarks

For efficient network performance a larger buffer size than the minimum discussed and presented in a table in this chapter should be chosen. Chapter *22 Traffic modelling of an input buffered switch* treats buffer utilisation based on traffic considerations. A buffer size should be chosen based on the "good" buffer size in this chapter and the buffer size defined in chapter *22 Traffic modelling of an input buffered switch*.

Appendix I:
Traffic modelling of an input buffered switch

In this appendix we will discuss some input buffered and some output buffered switch architectures. The SWIPP switch, which is input buffered, has the advantage of a simple architecture making an implementation of small physical size possible. We find that for some traffic patterns we have to pay for the simple architecture with reduced throughput. Increased capacity (bandwidth) is an attractive way to increase throughput, instead of choosing a more complex switch architecture. The contents of this appendix are helpful to understand the limitations of the switch architecture and to decide buffer sizes from expected traffic load.

Tables with buffer utilisation values for different architectures and load levels will be given. These simulated values are necessary for selecting good buffer sizes. A short discussion of the significant increase in average waiting time due to the few long packets will also be given.

In the following different switch architectures and load patterns will be presented and discussed. The first to be presented is a theoretical switch architecture, which closely resembles the SWIPP switch architecture.

I.0.1 A theoretical switch architecture.

Figure I.1 shows the basic switch architecture. This switch model has the following characteristics:
• 16 input and 16 output channels,
• same transfer capacity of input and output channels as well as of the lines internal to the switch,
• The arbitration algorithm is fair i.e. over time all input channels have equal chance to use an output channel.
• The input buffers are unlimited.

I.0.2 The default traffic pattern.

When nothing else is specified, the traffic pattern described below is used:
• The destination distribution is almost equal for all input channels. All output channels are chosen with equal probability. The only exception is that an input channel will not transmit to its own output channel.
• The packets arrive with a constant probability λ in each infinitely small time unit (i.e. the

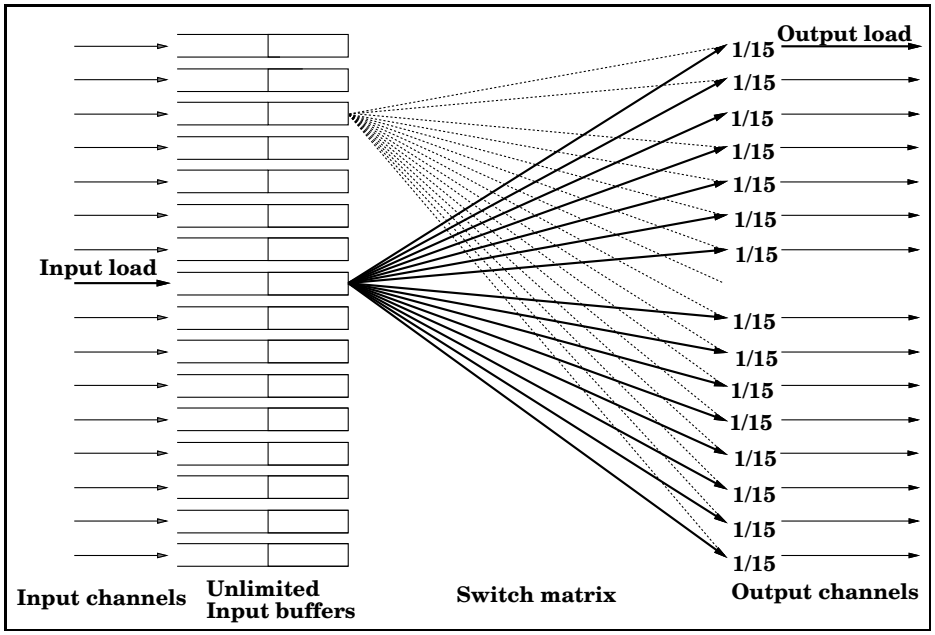


Figure I.1: The theoretical switch model used for comparison between switch architectures.

arrival process is described as a specific Markov source.) The probabilities for the different intervals between packet arrivals (the **p**robability **d**ensity **f**unction = pdf) follow a function:

$$\text{pdf}_m \cdot \Delta t = \lambda \Delta t (1 - \lambda \Delta t)^N \text{ where } \Delta t = \frac{t}{N} \tag{I.1}$$

$$\text{pdf}_m = \lim_{N \to \infty} \lambda (1 - \frac{\lambda t}{N})^N = \lambda e^{-\lambda t} \tag{I.2}$$

The average interval between packet arrivals is $\overline{t} = 1/\lambda$. The time t is divided into N small intervals of duration Δt.

• The time to transmit a packet through the cross section of a channel is also expected to have a similar behaviour[6]. While a packet is passing, there is an equal probability μ in each infinitesimal time unit that the packet shall end. This results in a probability function for packet lengths (pdf) equal to $\mu e^{-\mu x}$. The transmission time named x, has an average length $\overline{x} = 1/\mu$. In analytic statistics the packet length function corresponds to the service function. The transmission time of each individual packet is a function of the packet length L_{pk} and the channel transfer capacity C given by the equation: $x = L_{pk}/C$. L_{pk} is the packet length given in number of bits while the capacity C is the number of bits transmitted pr. time unit. Since x is a function of L_{pk} and C is constant the distribution of packet lengths L_{pk} follow the function $\frac{\mu}{C} e^{-\mu L_{pk}/C}$.

I.0.3 Load and saturation

Two important factors when we analyse switch architectures are the input load ρ_{in} and the output load ρ_{out}. The load is the part of the time a channel is busy transmitting. The load will be in the range $[0:1]$. The input load is the load on each input channel. ρ_{in} can be expressed as $\overline{x}/\overline{t} = \overline{x}\lambda$. The output load is the part of the time an output channel is busy forwarding data. The latter is also known as throughput and is in the following found with simulation.

[6] The time for the signal to propagate along the channel or delays due to queuing is not included.

The output load is equal to or less than the input load. When the switch can not get rid of packets with the same speed as they arrive the output load will be less than the input load. In this case the amount of packets in the unlimited input buffers will be constantly increasing. The waiting time will also be constantly increasing and no average waiting time can be found. In this situation the switch is in saturation. When the output load is equal to the input load the switch is below saturation. For this situation we can find average values for the waiting time and for the amount of packets in the input buffers.

$\rho_{out} = \rho_{in}$	below saturation	There is an average waiting time which can be found
$\rho_{out} < \rho_{in}$	in saturation	There is no average waiting time

I.1 Background: The D/D/1, M/M/1 and M/D/1 systems.

I.1.1 Deterministic and exponential distributions.

The default distribution both of the time intervals between packet arrivals and of the time to transmit packets are given by exponential distributions. In the following a short presentation of the exponential distribution generated by our Markov source and the distribution for a deterministic source is given. Some simulations with fixed packet lengths will also be discussed.

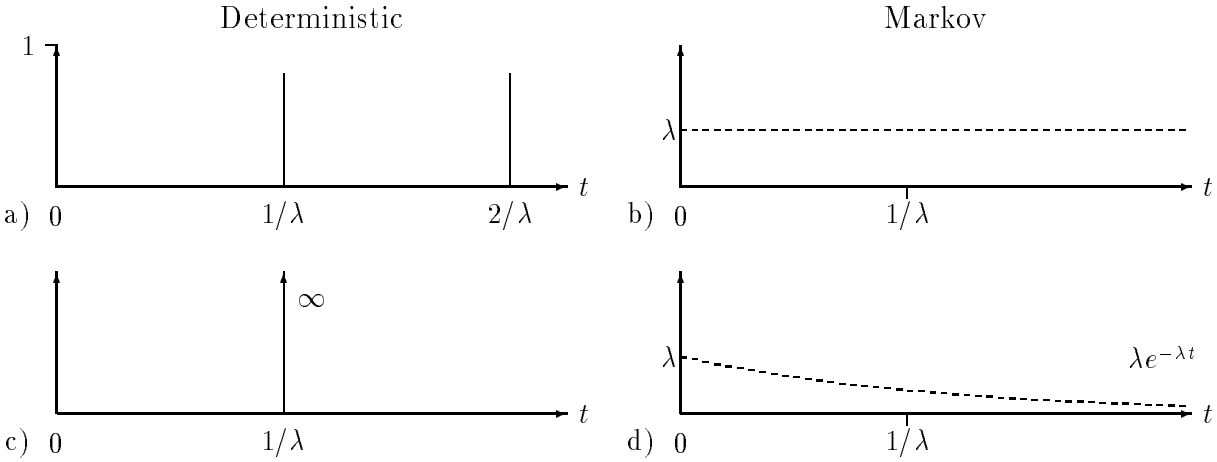


Figure I.2: Probability distribution and probability density for a deterministic and a Markov source.

The t=0 in figure I.2 is the arrival time of the last packet. a) and b) show the probability of a new packet arrival a time t after the last arrival. In the deterministic case the probability will be 1 at a time n/λ and zero between these values. n represents all natural numbers. For a Markov source the probability will always be equal to λ independently of the time since the last arrival. c) and d) show the probability of the *first* packet to arrive a time t after the last packet. Also in this case the probability will be equal to one at $1/\lambda$ for the deterministic source. The distribution for the Markov function will be the exponential function found earlier.

Both the deterministic and Markov source have an average time between packet arrivals equal to $1/\lambda$.

I.1.2 The A/B/m system.

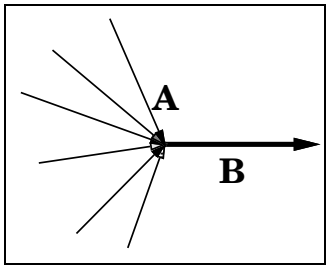


Figure I.3: Customers requesting for a resource.

A common syntax for analytic queuing theory is the **A/B/m** syntax. Generally the analytic system consists of a number of *customers* requesting for a *resource*. In a switch the resource is an output channel while the customers are the input channels which request that specific output channel.

A is the arrival function of the packets. **B** is the service function. B gives the time for which the customer is using the resource. This is the time a packet needs to be forwarded through a cross section of a channel. We name this time the transmission time. **m** is the resource capacity. For the following simulations m has been set to one. The utilisation ρ_{out} of the resource capacity is expressed as a number in the range $[0:1]$.

Examples of A and B functions are M (Markov as described above), D (deterministic as described above) and G (General i.e. any function).

I.1.3 Waiting and transmission time for the M/M/1 and D/D/1 systems.

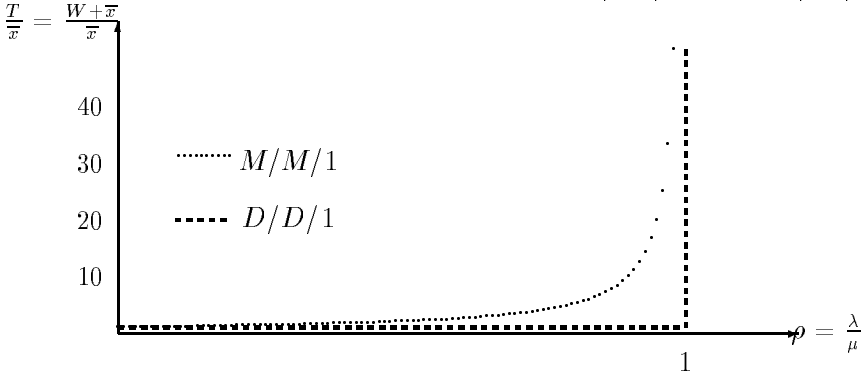


Figure I.4: Sum of waiting time and transmission time for D/D/1 and M/M/1. The curves represent normalised average values.

Figure I.4 shows the average sums of the waiting times and transmission times for the D/D/1 and M/M/1 systems. The resulting time has been normalised by dividing by the transmission time $\overline{x}$.

In the D/D/1 system both packet intervals and packet lengths will be fixed so there will be no queue (when $1/\lambda < 1/\mu$ i.e. $\rho < 1$). Thus the waiting time will be zero until the load reaches the system capacity.

In the M/M/1 system we have both variable intervals between packets and variable packet lengths. As the load decreases the waiting time will go against zero and the curve will approach one. As the load approaches the saturation point the waiting time will approach indefinity.

The time T we are studying is the sum of the time a packet has to wait, W and the time used to transmit the packet, x. T is normalised by dividing T with $\overline{x}$. The normalised value is denoted T_n. T_n for the M/M/1 system can be described ([61] eq. 1.26 and eq. 1.29) by the following equation:

$$T_n = T/\overline{x} = \frac{W + \overline{x}}{\overline{x}} = \frac{1}{1-\rho} = \frac{1}{1-\lambda/\mu} \tag{I.3}$$

I.1.4 The effect of changed capacity or load on the M/M/1 system.

In the following we will discuss how a change of channel and switch capacity will influence T i.e. the sum of the waiting time and transmission time. The packet length L_{pk} and arrival rate λ are unchanged. We use two capacity values C_{new} and C_{old} and two transmission times $\overline{x_{new}} = L_{pk}/C_{new}$ and $\overline{x_{old}} = L_{pk}/C_{old}$. We have a system with fixed $\overline{x_{old}}$, C_{old}, ρ_{old} and a T_{old} found from the previous three parameters. We will study how a change in transfer capacity C_{new} will influence T_{new}. Since we are looking for general relations C_{new} will always be discussed in relation to C_{old}. Similarly T_{new} will always occur together with T_{old}, on the form T_{new}/T_{old} or $T_{new}/\overline{x_{old}}$.

By manipulating equation I.3 we find the following relation:

$$\frac{T_{new}}{T_{old}} = \frac{1-\rho_{old}}{C_{new}/C_{old} - \rho_{old}} \tag{I.4}$$

We can also find the exact value for T_{new}

$$T_{new} = \frac{1}{C_{new}/C_{old} - \rho_{old}} \cdot \overline{x_{old}} \tag{I.5}$$

We see from equation I.4 that for small ρ_{old}, T_{new} is approximately linear with the inverse of C_{new}/C_{old} (when $C_{new}/C_{old} \gg \rho_{old}$. This is due to the reduction in x_{new} while the reduction of W_{new} is relatively small. For C_{new}/C_{old} larger than one the T_{new}/T_{old} relation will approach 0 as ρ_{old} approaches 1. Here the reduction will come from the reduction both in x_{new} and W_{new}. For C_{new}/C_{old} smaller than one the T_{new}/T_{old} relation will approach infinity as ρ_{old} approaches C_{new}/C_{old}. Also in this case the change will come from scaling of both x_{new} and W_{new}.

We may express T_{new} as a product of the scaling factor for the waiting time W_s and the scaling factor for the transmission time x_s. Thus we will have the following equation: $T_{new} = W_s x_s T_{old}$. The scaling factor for the transmission time is given by:

$$x_s = \frac{x_{new}}{x_{old}} = \frac{1}{C_{new}/C_{old}} \tag{I.6}$$

T_{new}						
C_{new}/C_{old}	$\rho_{old} = 0.1$		$\rho_{old} = 0.5$		$\rho_{old} = 0.9$	
0.25	$6T_{old}$	$\frac{20}{3}x_{old}$	∞	∞	∞	∞
0.50	$\frac{9}{4}T_{old}$	$\frac{10}{4}x_{old}$	∞	∞	∞	∞
0.75	$\frac{18}{13}T_{old}$	$\frac{20}{13}x_{old}$	$2T_{old}$	$4x_{old}$	∞	∞
1	T_{old}	$\frac{10}{9}x_{old}$	T_{old}	$2x_{old}$	T_{old}	$10x_{old}$
2	$\frac{9}{19}T_{old}$	$\frac{10}{19}x_{old}$	$\frac{1}{3}T_{old}$	$\frac{2}{3}x_{old}$	$\frac{1}{11}T_{old}$	$\frac{10}{11}x_{old}$
3	$\frac{9}{29}T_{old}$	$\frac{10}{29}x_{old}$	$\frac{1}{5}T_{old}$	$\frac{2}{5}x_{old}$	$\frac{1}{21}T_{old}$	$\frac{10}{21}x_{old}$
4	$\frac{9}{39}T_{old}$	$\frac{10}{39}x_{old}$	$\frac{1}{7}T_{old}$	$\frac{2}{7}x_{old}$	$\frac{1}{31}T_{old}$	$\frac{10}{31}x_{old}$
5	$\frac{9}{49}T_{old}$	$\frac{10}{49}x_{old}$	$\frac{1}{9}T_{old}$	$\frac{2}{9}x_{old}$	$\frac{1}{41}T_{old}$	$\frac{10}{41}x_{old}$
10	$\frac{9}{99}T_{old}$	$\frac{10}{99}x_{old}$	$\frac{1}{19}T_{old}$	$\frac{2}{19}x_{old}$	$\frac{1}{91}T_{old}$	$\frac{10}{91}x_{old}$

Table I.1: How a change of C influences T

The scaling factor for the waiting time can be expressed as:

$$W_s = \frac{W_{new}}{W_{old}} = \frac{(C_{new}/C_{old})(1-\rho_{old})}{C_{new}/C_{old} - \rho_{old}} \tag{I.7}$$

T_{new} may be expressed from the scaling factors in the following way:

$$T_{new} = W_{new} + x_{new} = W_s W_{old} + x_s x_{old} = W_s x_s T_{old} \tag{I.8}$$

In table I.1 there are two columns for each ρ_{old}. In one column T_{new} is expressed relative to T_{old} while the other column shows T_{new} relative to x_{old}.

We see from the table above that for small load the reduction in T_{new} as C_{new} increases is mainly due to the reduction in x_{new}. We remember that x_{new} is inversely proportional to C_{new} according to $x_{new} = L_{pk}/C_{new}$. The waiting time has little influence ($W_s \approx 1$).

For large loads both x_{new} and the waiting time decrease with increasing C_{new}. We can take $\rho_{old} = 0.9$ and $C_{new}/C_{old} = 10$ as an example. We read the value $T_{new} = 1/91\ T_{old}$ from the table above. We find from equation I.6 that $x_s = 1/10$ and from equation I.7 that $W_s = 10/91$. So in this case both the transmission time and the waiting time are reduced.

As will be demonstrated later in this appendix some switch architectures manage high load worse than other architectures while they perform similarly at more moderate load levels. If we have a simple switch and expect high load, we consider replacing the switch with another with better performance or increasing the channel capacity. The discussion above shows that increase in channel capacity efficiently reduces load, time for packet in switch, waiting time and buffer usage. This reduction is larger the closer the initial load is expected to be to the saturation point. The capacity may be increased at the switch architectural level by increasing the serial capacity (clock rate) or/and by increasing the parallel capacity (bus width). At the topological

level the capacity may be increased by adding more channels or switches in parallel. Thus when it is easy to add more switches and channels in parallel to reduce load bottlenecks it may be a cost-efficient solution to use simple switches.

I.1.5 The M/M/1 and M/D/1 models contra real networks.

The M/M/1 and M/D/1 systems as described above are easier to evaluate than real networks. They are based on an infinite number of sources generating packets at rates and time points completely independent with it self and with other sources. Thus the forwarding of one packet will not influence on generation or transmission of new packets from the same source, not even during transmission of the former. For the M/M/1 system packet lengths are independent of packet lengths from the same and other sources. Naturally these conditions are not true in real systems. In spite of this lack of accuracy, evaluation of the M/M/1 and M/D/1 systems give an approximate indication of how a real system may behave. Thus they give good initial values until measurement on real implementations can be done.

The arrival time of packets from a Markov source. .

The Markov source models a system where packets arrive with constant probability, independently of one another. This may look like a possible model for a system with a large number of sources running a number of different independent applications. It is expected to be less accurate when the sources are few, the applications few or when there is a high degree of synchronism between the time points of packet generation.

The Markov source generates the end of a packet.

In general networks running different applications simultaneously there will often be a high number of short packets while the longer packets are rare. Thus e^{-x} is a possible function to express the probabilities of different packet lengths

The deterministic (fixed) packet lengths.

For some applications fixed packet lengths are a natural choice. This may be the case where live sound or pictures are transmitted.

Conclusion

It is difficult to predict the traffic pattern for a network not built and for applications not defined. M/M/1 and M/D/1 are used as starting assumptions. There are two reasons for this choice:
•They are easier to simulate and calculate from than most other functions.
•Experience has shown that their behaviour is close enough to the traffic pattern in general networks so that good proposals can be made about network design.

In the following we will to some extent compare the M/M/1 and M/D/1 systems with the simulated behaviour of the input queues and/or the output queues.

Input Load	Output Load	Waiting Time
ρ_{in}	ρ_{out}	W_n
0.1	0.1	0.14
0.2	0.2	0.41
0.3	0.3	1.1
0.4	0.4	3.1
0.5	0.5	37.8
0.6	0.52	*362.6*
0.7	0.52	*568.0*
0.8	0.52	*662.7*
0.9	0.52	*714.6*
1.0	0.52	*734.0*

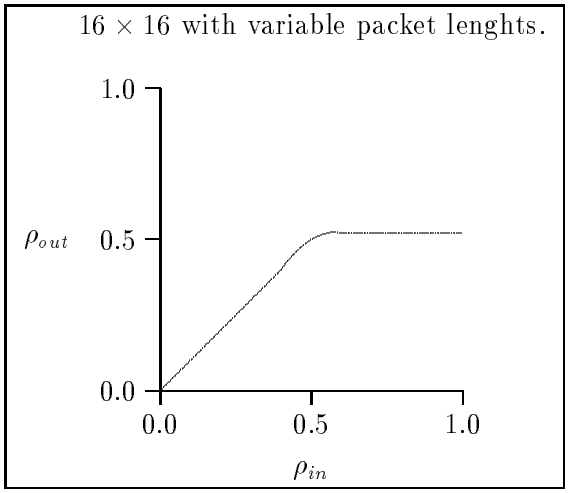


Figure I.5: Simulated performance values for a 16×16 *input buffered switch with variable packet length.*

I.2 The basic switch

Simulation results are presented in the following. The simulations have been stopped when the results have approximated a value. The number of decimals is intended to give an impression of the accuracy of the results. Numbers not approxing a specific value are written in *italic*. The value of these numbers depends on the time the simulations are terminated.

The table in I.5 shows the simulation results for a 16×16 switch. The waiting time W_n is given relative to $\overline{x}$.

We see that the output load follows the input load until the input load reaches approximately 0.52. At this point the switch can not carry away incoming packets fast enough. Above this point the input queue will continue to increase without limitation. When the queue increases, so will the waiting time. These unstable snapshot waiting times have been written in *italic* fonts in the table above.

I.2.1 Requirements on buffer sizes.

The values of the normalised waiting time given above is also the average number of packets which have to wait for service. If the input channels are busy receiving new packets 40% of the time, on average 3.1 packets will be stored in the input buffer. Thus if we want to make this storing local the input buffers must store at least 3.1 packets. If the buffers are smaller the storing will have to take place in the previous switch and the local bottle neck will also influence on the performance of the previous switch. If the input load increases to 50% the input buffers will need to store at least 37.8 packets to make the contention local.[7]

[7] Notice that these are average values. The buffer values covering 95% of the packet queues should be found. Anyway the average values give a good indication of these sizes.

I.2.2 Saturation due to the Head-Of-Line (HOL) contention.

We see from the table above that the switch saturates at an input load of 0.52. Hence 48% of the capacity of the output channels will not be used. The reduction in performance comes from the Head-Of-Line (HOL) contention: A packet to an available output channel can not be forwarded because it is blocked behind a packet to a busy output channel. The saturation value is a function both of the switch architecture and the destination distribution.

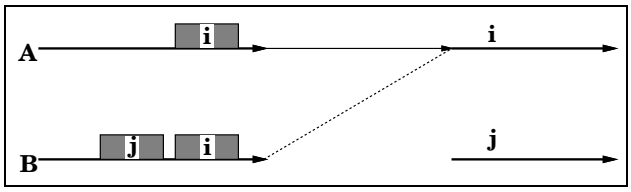


Figure I.6: Saturation due to Head-Of-Line contention.

Figure I.6 illustrates saturation due to the Head-Of-Line problem. Both the channel **A** queue and the channel **B** queue have a packet to output channel **i** in front. Channel **A** can transmit while channel **B** has to wait. Channel **B** has a packet to output channel **j** behind the blocked packet.

Other architectures which increase the saturation point will be discussed later.

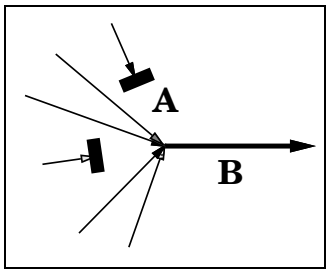


Figure I.7: Head-Of-Line may be illustrated as M/M/1 and M/D/1 systems where some of the sources are inhibited from delivering their service request to the system.

The dependency of the saturation point on the destination distribution.

In the distribution that we are using all input channels will select between the output channels with equal probability (figure I.8 **a)**). The only exception is that no input channel can select its own output channel. This distribution with the lowest saturation point is the one most clearly amplifying the difference between the characteristics of the different switch architectures. Figure I.8 **b)** shows a distribution where all input channels select between a smaller common set of channels. This distribution will have a higher saturation limit than the distribution given in **a)**. If one of the connected nodes has a central function like being a server, being a parent process for several child processes, or being a destination process for data collection, the distribution may be narrow-banded. As the bandwidth of the destination distribution is decreasing the difference in performance between switch architectures will be smaller. If the input channels have their

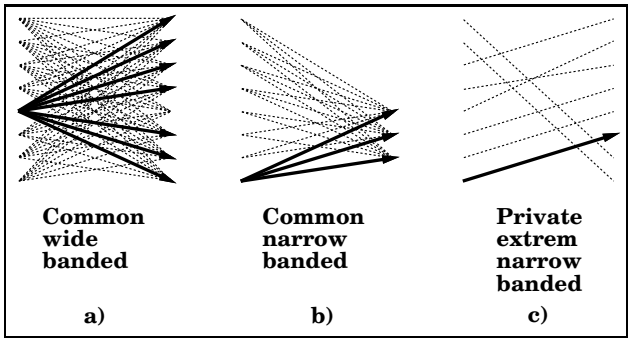


Figure I.8: Some destination distributions.

Input Load ρ_{in}	Output Load ρ_{out}		Waiting Time W_n	
	M Variable	D Fixed	M Variable	D Fixed
0.1	0.1	0.1	0.14	0.06
0.2	0.2	0.2	0.41	0.17
0.3	0.3	0.3	1.1	0.36
0.4	0.4	0.4	3.1	0.83
0.5	0.5	0.5	37.8	1.92
0.6	0.52	0.56	*362.6*	*14.89*
0.7	0.52	0.61	*568.0*	*91.75*
0.8	0.52	0.62	*662.7*	*150*
0.9	0.52	0.60	*714.6*	*178*
1.0	0.52	0.61	*734.0*	*195*

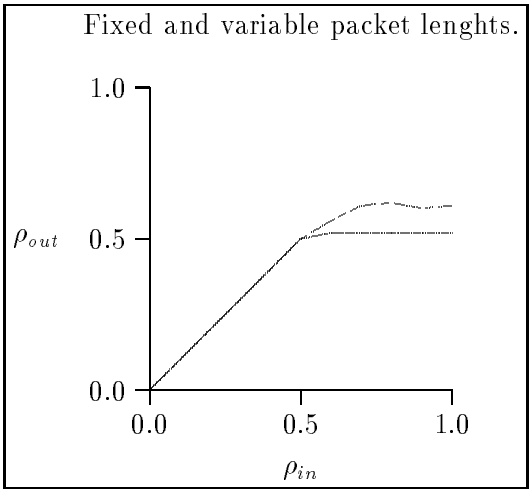


Figure I.9: Simulated performance values for a 16×16 *input buffered switch with fixed and variable packet lengths.*

private set of preferred output channels also the saturation limit will increase. In the extreme case, when each input channel has its own private output channel not shared with any other input channels, the saturation limit will be 1.0 (fig. I.8 **c)**). Private connections may be the pattern in parallel processing systems where all processors are accessing their "next neighbour" and all processors have different neighbours.

The uniform common destination distribution (fig. I.8 **a)**) was chosen in the modelling since it gave the lowest saturation point and the largest difference in performance between switch architectures.

I.2.3 Fixed and variable packet lengths

The variation of packet lengths influences on the critical load and the waiting time. Table I.9 shows some simulation results where variable packet lengths (named M for Markov) are compared with fixed packet lengths (named D for deterministic). The average packet lengths are equal.

The simulation results show that for the same average waiting time fixed packet lengths will

both support higher output load (throughput) and have a shorter waiting time than for variable packet lengths. For fixed packet lengths the saturation value is approximately 0.60 compared to approximately 0.52 for variable packet lengths.

We notice that fixed packet lengths give shorter waiting time than variable lengths even when the average lengths are equal. We will in the following briefly discuss the waiting time for a M/M/1 (variable packet length) system contra a M/D/1 (fixed packet length) system not suffering from the Head-Of-Line contention. For our switch the influence of the Head-Of-Line effect is least for small load.

I.2.4 Expected remaining waiting time for a M/M/1 and a M/D/1 system.

From [61] eq. 1.82 we have that the expected time a newly arriving customer must spend in the queue while the customer (if any) which he finds in service completes his remaining required service time is $W_0 = \lambda\overline{x^2}/2$. If we calculate this waiting time for the fixed and variable packet lengths we find the waiting time for the fixed length to be $W_0(D) = \lambda\frac{1}{2}\frac{1}{\mu^2}$ while it for the variable length is $W_0(M) = \lambda\frac{1}{2}\frac{2}{\mu^2}$ i.e. the waiting time for variable packet length is twice that of the fixed packet lenght. This difference is caused by the longest packets. Although the long packets are few their contribution to the average waiting time is considerable. We find that this result corresponds to the simulation results given in our table above for small loads. As the load increases there will be a growing probability that a waiting packet also waits for other packets than the one in service. Also the Head-Of-Line saturation will be significant. Both make the difference in waiting time and output load between fixed and variable packet lengths grow even faster.

It will be natural from these considerations to ask whether we could choose a maximum packet length when variable packet lengths are allowed. We could select this maximum length for the variable packet so that the average waiting time for the packet in service is equal for the variable and fixed packet sizes. In the following we calculate an approximate packet length.

I.2.5 The influence of long packets on the average waiting time.

The probability density function (pdf) for the Markov function is given by $m(x) = \mu e^{-\mu x}$ while the pdf for the fixed (deterministic) function is given by $d(x) = u_0(x - \frac{1}{\mu})$. Here u_0 is the unit impulse function which has a value of ∞ when $x - \frac{1}{\mu}$ are equal to 0. For all other values u_0 is equal to 0.

First, according to [61] eq. 1.82, we must find the expected value of x^2. This is in general:

$$E[x^2] = \int_0^\infty x^2 \text{ pdf } dx$$

For the fixed packet size this is:

$$E_d[x^2] = \int_0^\infty x^2 u_0(x - \frac{1}{\mu})dx = \frac{1}{\mu^2}$$

For the variable packet length we will set the upper limit to z. Thus we can not use the exponential function found before. To get some knowledge about variable packet lengths with an upper limit we create a new pdf with the following requirements:

$$\text{pdf}_v(0) = \mu,$$
$$\text{pdf}_v(z) = 0,$$
$$\int_0^z (pdf)_v dx = 1, \text{ and}$$
$$\lim_{z\to\infty} \text{pdf}_v = \mu e^{-\mu x}.$$
$$\text{pdf}_v \text{ shall have a shape close to } \mu e^{-\mu x}.$$

We chose the following function:

$$\text{pdf}_v = \frac{\mu e^{-\mu x} - \mu e^{-\mu z}}{1 - e^{-\mu z}} = \frac{\mu e^{-\mu x}}{1 - e^{-\mu z}} - \frac{\mu e^{-\mu z}}{1 - e^{\mu z}} = \frac{\mu e^{-\mu x}}{1 - e^{-\mu z}} - C_0 \tag{I.9}$$

Then we find $E_v[x^2]$ for this pdf-function.

$$E_v[x^2] = \int_0^z x^2 \text{pdf}_v dx = e^{-\mu z}(z^2 + 2z\frac{1}{\mu} + \frac{1}{\mu^2}) = e^{-\mu z}(z + \frac{1}{\mu})^2 \tag{I.10}$$

When we set $E_d[x^2] = E_v[x^2]$ we get:

$$\frac{1}{\mu^2} = e^{-\mu z}(z + \frac{1}{\mu})^2$$

By iteration we find $\underline{\underline{z \approx 2.5\frac{1}{\mu}}}$.

Notice that the average packet length for pdf_v is not $1/\mu$ but $(1 - e^{-\mu z})\frac{1}{\mu} = 0.92\frac{1}{\mu}$.

If we integrate μe^{-mux} from 0 to z we get an idea of the fraction of packets in a totally unlimited packet length distribution that are shorter than z. This will not give an exact answer since z found for one pdf is used for another pdf, but other calculations indicate that the error will be less than 1 %.

With this limitation the quota of packets smaller than z is :

$$\int_0^z \mu e^{-\mu x} dx = \Big[-e^{-\mu x} \Big]_0^z = e^0 - e^{-\mu z} = \underline{\underline{0.92}}$$

The 92% of the packets that have a packet length shorter than 2.5 average packet lengths have an average waiting time similar to the packets with fixed length. The remaining 8 % having a packet length longer than 2.5 average packet lengths double the average waiting time.[8]

Use of maximum limit for variable packet lengths

If it is important to get the longest packets through the network as quickly as possible, a maximum packet length can not be chosen. If smaller packets have higher priority, the packets over a maximum value must be rejected or transmitted in parts, with a time interval between them. Thus the application will decide whether a maximum limit should be put to packet length.

[8] This result is the work of the author.

Input Load ρ_{in}	Output Load ρ_{out}					Waiting Time W_n					
	$2\times 2^{\dagger}$	$4\times 4^{\dagger}$	$8\times 8^{\dagger}$	$16\times 16^{\dagger}$	$32\times 32^{\dagger}$	1×1	$2\times 2^{\dagger}$	$4\times 4^{\dagger}$	$8\times 8^{\dagger}$	$16\times 16^{\dagger}$	$32\times 32^{\dagger}$
0.1	0.1	0.1	0.1	0.1	0.1	0.11	0.12	0.16	0.14	0.14	0.14
0.2	0.2	0.2	0.2	0.2	0.2	0.25	0.26	0.38	0.43	0.41	0.42
0.3	0.3	0.3	0.3	0.3	0.3	0.43	0.43	0.92	0.95	1.1	1.1
0.4	0.4	0.4	0.4	0.4	0.4	0.67	0.65	2.0	2.5	3.1	3.6
0.5	0.5	0.5	0.5	0.5	0.5	1.00	1.0	4.6	14.7	37.8	21.2
0.6	0.6	0.6	0.54	0.52	0.51	1.50	1.6	33.2	*101.3*	*362.6*	*75.0*
0.7	0.7	0.62	0.55	0.52	0.51	2.33	2.4	*113.7*	*165.7*	*568.0*	*107.2*
0.8	0.8	0.62	0.55	0.53	0.51	4.00	4.5	*160.1*	*203.1*	*662.7*	*117.3*
0.9	0.9	0.62	0.55	0.52	0.51	9.00	8.8	*186.5*	*224.6*	*714.6*	*126.6*
1.0	1.0	0.62	0.55	0.52	0.51	∞	*24.2*	*205.3*	*229.4*	*734.0*	*125.1*

Table I.2: Simulated output load and waiting time for some switch sizes.

I.3 Other switch architectures: switch sizing

We have chosen a switch with 16 input and 16 output channels. A natural question is whether this is a clever choice. We will therefore in the following look at the saturation values and waiting times for some switch sizes and input loads.

Table I.2 shows output load and waiting time for some switch sizes. Observe that for all $n \times n$ columns marked by a dagger (†) each input channel only switches between $n-1$ of the output channels (input channel i does not switch to output channel i). The simulation for the 32×32 switch did not run for as long time as the simulations for the other switch sizes. Thus the waiting times for the 32×32 switch are not directly comparable to other switch sizes in the saturated region.

The leftmost column for the waiting time gives analytically calculated values. These values should be equal to the values for the 2×2 switch. The difference between these two columns can give an impression of the accuracy of the simulations.

We find from table I.2 that the saturation limit is approximately 0.55 ± 0.05 for all switches larger than the 4×4 channel switch. Thus it makes no large difference whether we choose an 8×8, 16×16 or 32×32 channel switch when it comes to saturation. The difference in waiting time is not large except for the values closest to the saturation limit. We can conclude from the figures above that our switch size for the parameters evaluated is approximately an equal choice as the 8×8 and 32×32 alternatives.

I.4 Other switch architectures: Queue manipulation

The knowledge about the Head-Of-Line problem naturally leads to the idea to bypass packets in the input queues. If the first packet in the input queue is blocked the second packet could be checked. In this way the input queue could be searched backwards either towards the end of the queue or until a packet addressed to an available output channel is found. If we could always search all the way to the end of the input queue we would have eliminated the Head-Of-Line effect.

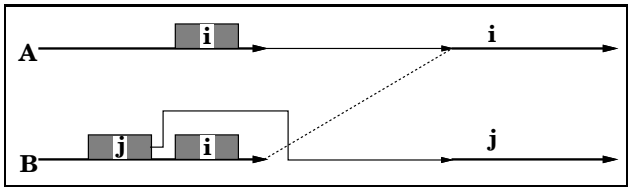


Figure I.10: Bypassing used to reduce the Head-Of-Line effect.

Variable packet lengths												
Input Load ρ_{in}	Output Load ρ_{out}						Waiting Time W_n					
Bypassed:	0	1	3	7	15	31	0	1	3	7	15	31
0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.14	0.13	0.13	0.13	0.13	0.13
0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.41	0.32	0.31	0.30	0.31	0.31
0.3	0.3	0.3	0.3	0.3	0.3	0.3	1.1	0.7	0.6	0.6	0.6	0.6
0.4	0.4	0.4	0.4	0.4	0.4	0.4	3.1	1.3	1.0	1.0	1.0	1.0
0.5	0.5	0.5	0.5	0.5	0.5	0.5	37.8	3.1	1.8	1.8	1.8	1.8
0.6	0.52	0.6	0.6	0.6	0.6	0.6	*362.6*	9.9	3.4	3.0	3.0	3.0
0.7	0.52	0.65	0.7	0.7	0.7	0.7	*568.0*	*170.2*	8.4	4.6	4.7	4.7
0.8	0.53	0.67	0.76	0.8	0.8	0.8	*662.7*	*368.3*	*41.0*	10.5	7.6	7.9
0.9	0.52	0.67	0.77	0.85	0.9	0.9	*714.6*	*463.1*	*95.0*	*38.5*	25.0	17.1
1.0	0.52	0.68	0.76	0.85	0.92	0.94	*734.0*	*527.9*	*129.0*	*77.0*	*53.0*	*45.7*

Figure I.11: Simulated output load and waiting time for a switch with bypass possibility. If the packets in front of the queue are headed for busy output channels, packets behind may bypass them. The bypass values in the upper digit row tells how many packets which may be bypassed.

The table below shows the simulation results for a switch with bypass in the input queues. The numbers 0, 1, 3, 7, 15 and 31 are the maximum number of packets which are passed in a search for a packet addressed to an available output channel. Thus the zero columns are the known values for the switch without bypassing.

We see from table I.11 that if we design the switch architecture so that the second packet is allowed to bypass the first one if it is blocked, the saturation load will increase with approximately 26%. If we instead of bypassing one packet allow bypassing of up to three packets the saturation load will increase with 43%. As we could expect we will gain more by allowing bypass of the first packets than the later ones. In real implementations there will be a limit for how much we are interested in paying for the last and small increase in saturation.

Increase of output load by using bypassing demands that the destination distribution has a certain width (figure I.8). If the destination distribution is narrow-banded or private the increase in performance will be smaller.

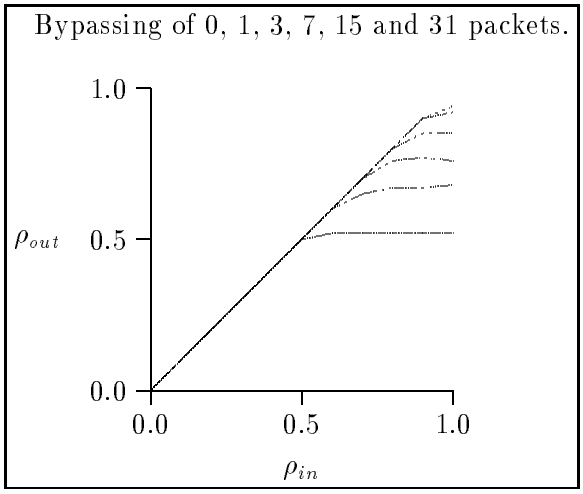


Figure I.12: The output load for different maximum bypass values. The bypass values are increasing upwards.

Fixed packet length.

It is easier to implement bypassing for input queues with fixed than with variable packet lengths. The table below shows the output load and waiting time when 0, 1 and 7 packets in the input queue can be bypassed in search for a packet addressed to an available output channel.

Fixed packet length						
Input Load ρ_{in}	Output Load ρ_{out}			Waiting Time W_n		
Bypassed:	0	1	7	0	1	7
0.1	0.1	0.1	0.1	0.06	0.06	0.06
0.2	0.2	0.2	0.2	0.17	0.14	0.14
0.3	0.3	0.3	0.3	0.36	0.29	0.28
0.4	0.4	0.4	0.4	0.83	0.50	0.48
0.5	0.5	0.5	0.5	1.92	1.05	0.81
0.6	0.56	0.6	0.6	14.9	1.9	1.4
0.7	0.61	0.7	0.7	91.8	6.1	2.3
0.8	0.62	0.73	0.79	150	55	4.95
0.9	0.60	0.63	0.87	178	101	15.4
1.0	0.61	0.74	0.89	195	129	52.8

By comparing fixed and variable packet length we find that fixed packet length still support a somewhat higher output load than the variable. Not surprisingly the difference gets smaller as the output load approaches 1.0. We also see that below saturation the average waiting time (= average used buffer space) used with fixed packet length is approximately half of the average waiting time with variable packet lengths.

Example:
We have an approximate input load of 0.7. With a variable packet length the switch would be in saturation and the waiting time would be constantly increasing. If we instead had a fixed packet

size we would still be in saturation but the waiting time would be less than 1/4. If the input buffer could queue two packets and allow bypassing of the first the waiting time would drop down to approximately $6 \cdot \overline{x} = 6L_{pk}/C$. If the buffer size was increased to more than 6 packet lengths the contention would be mostly local.

I.5 Other switch architectures: Output buffering.

As we have seen the Head-Of-Line effect reduces the performance of the switch considerably. We have already studied how bypassing can be used in the input queues to reduce this effect. Another way to avoid this performance degradation is always to forward the packets directly to the right output channel. In this architecture an output channel may receive a number of packets simultaneously but can forward only one at a time. Therefore buffering is needed.

The output buffer architectures have the following characteristics in addition to buffering at the output side of the switch matrix:
- No local Head-Of-Line contention.
- The M/M/1 (variable packet length) and M/D/1 (fixed packet length) models are approximately correct.
- No arbitration at the input channels.
- More complex switch matrix and
- More difficult to implement flow control than for the input buffered switch.

The switch matrix of these switches need a much larger capacity since a number of packets can be switched simultaneously to the same output channel. The capacity may be increased by forwarding the packets at higher speed serially through the switch matrix. In this case some buffering is required at the input channels. Another alternative is through increased parallelity by letting each input channel have a dedicated bus to each of the output channels. This is a lot of wiring and is difficult to implement on one switch circuit.

It is difficult to implement a flow control scheme for output buffered switches, and many switches have been implemented without. For those without flow control packets have to be dismissed when the output channels run out of buffer space. Thus it is necessary to know about the traffic pattern so that the buffer capacities are large enough to make the packet loss rate acceptably small.

We will in the following study the performance of some of these architectures. It should also be noted that in systems with simple flow control a saturation similar to Head-Of-Line occurs between output buffered switches.

The multiple plane switch.

Figure I.14 shows a drawing of a multiple plane switch. Here first input channel **A** is forwarding a packet to output channel **i**. Input channel **B**, which also has a packet addressed to output channel **i** finds this channel busy. Input channel **B** thus advances to the next switch level and tries to establish a connection here. A switch with a large number of switch planes has been simulated.

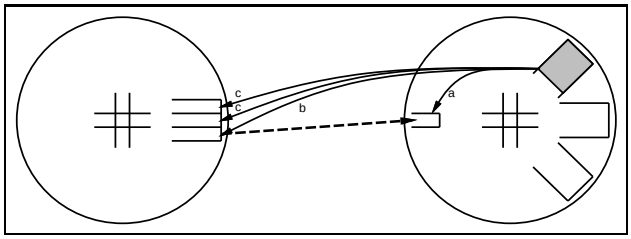


Figure I.13: Head-Of-Line buffering in output buffered switches with local flow control. The output buffer of the leftmost switch functions as an input buffer for the rightmost switch. If the packet in front is blocked, the remaining packets can not pass and we have a Head-Of-Line blockage. If the remaining packets are allowed to pass, we have a bypass buffer similar to the input bypass buffer discussed in the previous pages.

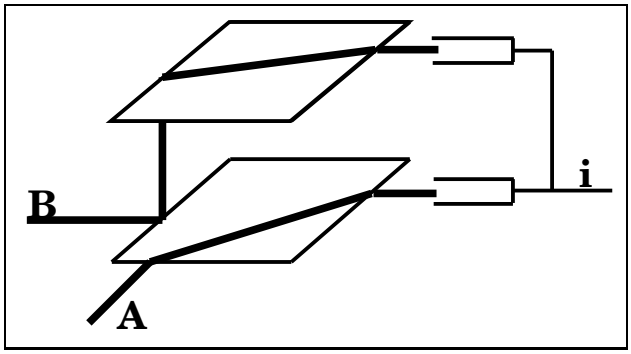


Figure I.14: Multiple planes.

Plane nr.:	1	2	3	4
Throughput/ρ_{out}	0.52	0.30	0.13	0.05

When we accumulate these values, we find that with two planes we have a throughput of approximately 0.82. By adding one more plane we increase the total throughput to 0.95. With a total of four planes the total throughput will exceed 99%.

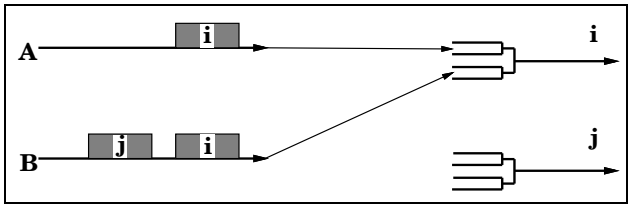


Figure I.15: Multiple output queues.

Another way to draw the output buffered switches is given in figure I.15. Here each input channel either has a private buffer at each output channel or requests one buffer from a buffer bank at each output channel.

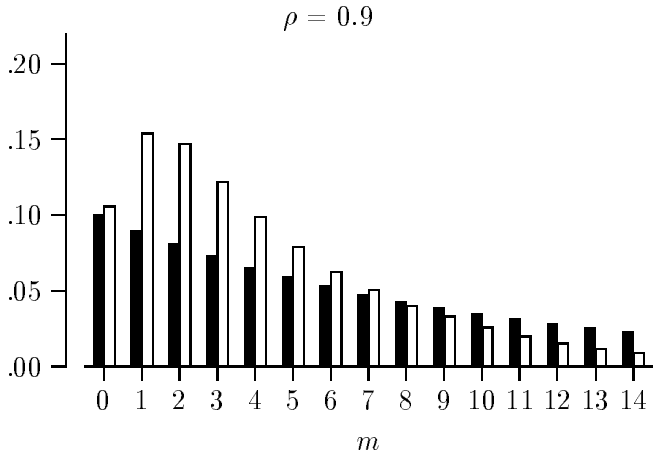
Load:	Number of packets in buffer										
	0	1	2	3	4	5	6	7	8	9	10
0.05	0.950	0.048	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	0.950	*0.047*	*0.002*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*
0.10	0.901	0.093	0.005	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	0.900	*0.090*	*0.009*	*0.001*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*
0.20	0.803	0.174	0.021	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	0.800	*0.160*	*0.032*	*0.006*	*0.001*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*
0.30	0.703	0.243	0.046	0.007	0.001	0.000	0.000	0.000	0.000	0.000	0.000
	0.700	*0.210*	*0.063*	*0.019*	*0.006*	*0.002*	*0.001*	*0.000*	*0.000*	*0.000*	*0.000*
0.40	0.601	0.295	0.083	0.017	0.003	0.000	0.000	0.000	0.000	0.000	0.000
	0.600	*0.240*	*0.096*	*0.038*	*0.015*	*0.006*	*0.002*	*0.001*	*0.000*	*0.000*	*0.000*
0.50	0.503	0.322	0.122	0.038	0.011	0.003	0.001	0.000	0.000	0.000	0.000
	0.500	*0.250*	*0.125*	*0.062*	*0.031*	*0.016*	*0.008*	*0.004*	*0.002*	*0.001*	*0.000*
0.60	0.401	0.328	0.161	0.064	0.028	0.012	0.004	0.002	0.001	0.000	0.000
	0.400	*0.240*	*0.144*	*0.086*	*0.052*	*0.031*	*0.019*	*0.011*	*0.007*	*0.004*	*0.002*
0.70	0.304	0.309	0.191	0.097	0.050	0.025	0.013	0.006	0.003	0.002	0.001
	0.300	*0.210*	*0.147*	*0.103*	*0.072*	*0.050*	*0.035*	*0.025*	*0.017*	*0.012*	*0.008*
0.80	0.205	0.255	0.200	0.129	0.080	0.052	0.031	0.018	0.012	0.007	0.004
	0.200	*0.160*	*0.128*	*0.102*	*0.082*	*0.066*	*0.052*	*0.042*	*0.034*	*0.027*	*0.021*
0.90	0.105	0.151	0.148	0.125	0.101	0.082	0.067	0.051	0.039	0.029	0.022
	0.100	*0.090*	*0.081*	*0.073*	*0.066*	*0.059*	*0.053*	*0.048*	*0.043*	*0.039*	*0.035*
0.95	0.054	0.084	0.090	0.084	0.073	0.067	0.059	0.051	0.046	0.040	0.035
	0.050	*0.048*	*0.045*	*0.043*	*0.041*	*0.039*	*0.037*	*0.035*	*0.033*	*0.032*	*0.030*

Table I.3: The probability for different numbers of packets in a buffer

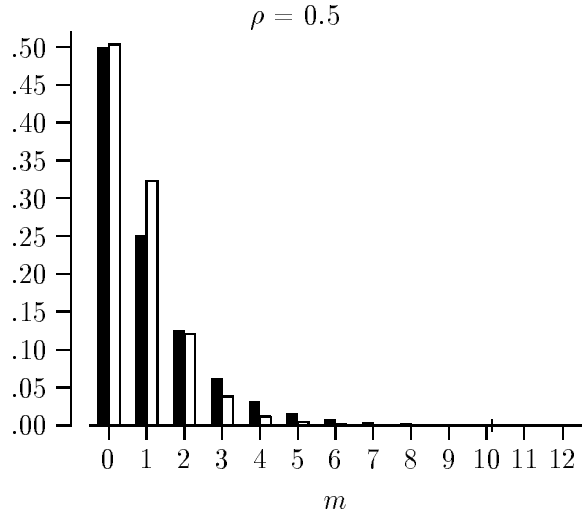
I.5.1 Buffer use and packet dismiss rate for output buffered systems.

Table I.3 shows some simulation and analytic results for the M/M/1 (variable packet length) and M/D/1 (fixed packet length) systems. The numbers of packets in a buffer are written in the horizontal direction. Different load amounts are given in the vertical direction. There are two rows for each load value. The upper rows are the simulated values for fixed packet sizes. These figures have been written in an ordinary vertical font. The lower rows written in *italic* are for variable packet sizes. The analytic function $(1-\rho)\rho^k$ ([61] eq. 1.56) is used to find the probabilities that the buffer contains k packets.

Systems with fixed packet sizes will on average use less buffer space than the systems with variable packet sizes. The average number of packets in a buffer is equal to: $\sum_{i=0}^{\infty} kp_k$. Based on the numbers in I.3, for a load of 0.5 the average number of fixed-length packets is 0.75 while the average number of variable-length packets is 1.0. For a load of 0.9 the average number of fixed-length packets is between 4.2 and 4.5 (p_k for the highest k's is missing) while for variable-length packets it is 9.0. The values for the variable packet lengts may be expressed by the equation $\rho/(1-\rho)$ (Eq. 1.57 in [61]). The distribution of (and weight) of p_k can be visualised by drawing histograms for some of the loads in table I.3.



The histogram above shows the probability distribution for the different numbers of packets waiting for transmission at a load 0.9. We can clearly see the difference in shapes between fixed packet length (white pillars) and variable packet lengths (black pillars).



This histogram shows the distribution of a waiting queue with a load 0.5. We also here have a significant difference between fixed and variable packet lengths for the probability of one packet in the system.

Dismiss rate.

p_i is the probability that an unlimited buffer contains i packets. We will now look at an output buffer with buffer space for k packets. Since a buffer will not contain more than k packets the probabilities $p_{k+1}, p_{k+2} \ldots p_\infty$ must be removed from the distribution and the probabilities $p_0, p_1, \ldots, p_{k-1}, p_k$ be scaled according to this. We name the probability that a buffer with a maximum size of k packets contain j packets P_j. For a M/M/1 system we have the following scaling:

$$P_j = \frac{p_j}{\sum_{i=0}^{k} p_i} \tag{I.11}$$

The probability that the buffer is full is P_k. The number of dismissed packets is the number of packets received while the buffer is full. $P_k T$ is the part of a time T where all buffers are in use. $\lambda P_k T$ is the number of dismissed packets during T. The total number of received packets during this time is λT. Thus we have the following equation:

$$P_{\text{dismissed}} = \frac{\text{packets received while full}}{\text{total number of received packets}} = \frac{\lambda T P_k}{\lambda T} = P_k = \frac{p_k}{\sum_{i=0}^{k} p_i} \qquad \text{(I.12)}$$

As stated above equation I.11 is strictly limited to the M/M/1 system. In the following it will also be used for the M/D/1 system. This gives a small error but should give an indication of how the dismiss rates increase and decrease relatively to other values.

The table below shows the dismiss rate for different buffer sizes at different load levels.

Load:	Number of packets in buffer										
	0	1	2	3	4	5	6	7	8	9	10
0.05	1.000	0.048	0.001	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	1.000	*0.048*	*0.002*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*
0.10	1.000	0.094	0.005	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	1.000	*0.091*	*0.009*	*0.001*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*
0.20	1.000	0.178	0.021	0.002	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	1.000	*0.167*	*0.032*	*0.006*	*0.001*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*	*0.000*
0.30	1.000	0.257	0.047	0.007	0.001	0.000	0.000	0.000	0.000	0.000	0.000
	1.000	*0.231*	*0.065*	*0.019*	*0.006*	*0.002*	*0.001*	*0.000*	*0.000*	*0.000*	*0.000*
0.40	1.000	0.329	0.084	0.017	0.003	0.000	0.000	0.000	0.000	0.000	0.000
	1.000	*0.286*	*0.103*	*0.039*	*0.016*	*0.006*	*0.002*	*0.001*	*0.000*	*0.000*	*0.000*
0.50	1.000	0.390	0.129	0.039	0.011	0.003	0.001	0.000	0.000	0.000	0.000
	1.000	*0.333*	*0.143*	*0.067*	*0.032*	*0.016*	*0.008*	*0.004*	*0.002*	*0.001*	*0.000*
0.60	1.000	0.450	0.181	0.067	0.028	0.012	0.004	0.002	0.001	0.000	0.000
	1.000	*0.375*	*0.184*	*0.099*	*0.056*	*0.033*	*0.019*	*0.011*	*0.007*	*0.004*	*0.002*
0.70	1.000	0.504	0.238	0.107	0.052	0.026	0.013	0.006	0.003	0.002	0.001
	1.000	*0.412*	*0.224*	*0.135*	*0.087*	*0.057*	*0.038*	*0.026*	*0.018*	*0.012*	*0.009*
0.80	1.000	0.553	0.303	0.164	0.092	0.056	0.032	0.018	0.013	0.007	0.004
	1.000	*0.444*	*0.262*	*0.173*	*0.122*	*0.089*	*0.066*	*0.050*	*0.039*	*0.030*	*0.023*
0.90	1.000	0.590	0.366	0.237	0.160	0.116	0.087	0.061	0.045	0.033	0.024
	1.000	*0.474*	*0.299*	*0.212*	*0.160*	*0.126*	*0.102*	*0.084*	*0.070*	*0.059*	*0.051*
0.95	1.000	0.607	0.393	0.269	0.189	0.148	0.115	0.091	0.075	0.062	0.051
	1.000	*0.487*	*0.316*	*0.231*	*0.180*	*0.146*	*0.122*	*0.104*	*0.090*	*0.079*	*0.069*

Examples:
If we have a buffer size for 6 packets and a load of 0.5 in a system with fixed packet lengths 0.1 % of the packets will be dismissed. In a similar system with variable packet lengths 0.8 % of the packets will be dismissed. To reduce the dismiss rate for variable packet lengths the buffer size must be increased to 9 packet lengths. If the load is 0.9 with a 10-packet-long buffer 2.4 % of the packets with fixed length will be dismissed. In a system with variable packet lengths 5.1 % of the packets will be dismissed.

I.6 Conclusion

We have studied the performance of the switch architecture chosen for SWIPP. We have estimated throughput and waiting time. These values can together with the channel capacity, packet length and expected load be used to find an estimated time for transmission of a packet from the source

to the destination. The tables can be used to give indications on how large the buffers should be to make the contention effects local. We have also noted the important increase in waiting time from the few very long packets in systems with variable packet lengths.

Some other switch architectures have also been presented which offer better performance than the SWIPP switch. This author's impression, based on literature, sketches and simulations not presented here is that these are much more complicated to implement. Thus the SWIPP switch is chosen as a trade-off between performance and architecture complexity.

Appendix J:
Discussion of clocking strategy

The choices of clocking strategy and clock rates are important topics influencing on system performance, power consumption, error rate, switch architecture and choice of materials for substrate. It influences on propagation time and bandwidth. Secure implementation/design (or the lack of it) influences on error rate. Present versions of the SWIPP circuits have been designed for high clock rates compared to standard digital circuit design, but without pushing the technologies to its limits. The reason for the choice of moderate clock rates is that design for high clock rates requires knowledge, experience and advanced test equipment not yet achieved by the SWIPP project members. So far, SWIPP architecture and system knowledge have been given higher priority. This appendix will discuss some of the considerations already made and considerations that must be made regarding clock strategies and clock rates. This appendix contains a mixture of textbook material and the author's own experiences.

As discussed earlier in this thesis, some of the factors that make a high performance network are low latency and high bandwidth. There are several ways to attain this goal. One way is by having a high clock rate, i.e. keeping data for as short time as possible in each location.

Making it possible to have fast clocking, depends on several factors:
1) A technology supporting a high enough clock speed: CMOS, BiCMOS, ECL or GaAs.
2) Fast storage elements and a robust clocking scheme.
3) Clock distribution with small skew.

This appendix deals with the last two factors above.

J.0.1 Storage elements and clock schemes.

There are mainly two clock schemes used in high-speed clocking:
— Single-phase clocks, and
— Pseudo-two-phase clocks.
Pseudo-two-phase clocks require two clock signals. Clock skew between these two lines has to be considered. Since single-phase clocks use only one clock signal the problem is not present for this clock strategy. Therefore single-phase clocks are often preferred at the highest clock rates.

J.1 Single-phase clock equations for proper operation.

In this clocking scheme clock skew will only appear due to differences in propagation time of the clock signal to different physical positions on the circuit.

For correct operation, analysis of propagation time through latches and logic is required. In this clocking scheme this analysis has to be performed a little more carefully than for comparable clocking schemes. In other clock schemes faulty operation may be avoided by changing externally fed clock signals. With this clock scheme we may have fault relations which are not so easily "repaired".

When correctly designed, this clocking scheme has the potential of reaching the highest clock speeds.

J.1.1 Relations to be fulfilled for proper operation.

As stated above, for correct operation the clock periods have to fulfil some requirements.

Sufficient time between clock edges for signal propagation.

First, the minimum time between two used clock edges has to be larger than the maximum signal propagation time from when a new value is released in a latch until the signal is captured in the succeeding latch. This maximum time has to include the worst possible clock skew between the clock at the "transmitting" and "receiving" ends. To simplify the calculation, the maximum path length may be replaced by the sum of the maximum of each individual element according to the rule: $max(x+y+z) \leq max(x)+max(y)+max(z)$. The relation to be fulfilled is expressed by the following inequality (based on [6] eq. 8.1):

$$min(t_{clock-cycle}) > max(t_{flip-flop}) + max(t_{logic}) + max(t_{setup}) + max(t_{clock-skew(R-T)}) \quad \text{(J.1)}$$

Here $t_{clock-cycle}$ is the time between two used clock edges. This means between positive (negative) clock edges if the logic contains only positive (negative) edge triggered latches. If the logic consists of dual edge triggered latches or the latches are arranged with every second latch as positive and negative edge triggered, the time interval is between a positive (negative) clock edge to the succeeding negative (positive) clock edge. The $t_{flip-flop}$ value is the time from a clock edge passes the 50% level until the latch output value released by the clock edge has propagated to the latch output. The time for a signal to propagate through the combinatory logic (latches excluded) is given by t_{logic}. The set-up time, t_{setup}, is the time before the clock edge a signal has to be present at the latch input to give correct interpretation and transfer to the latch output. The clock skew, $t_{clock-skew}$, is the point of time at which a clock edge reaches the receiving latch minus the point of time the same clock edge reaches the transmitting latch. Thus clock skew comes from differences in propagation time of the clock signal.

Signal stable long enough for correct capture.

The hold time, t_{hold}, is the amount of time for which the signal has to stay stable at the latch input, after the clock transition, to guarantee correct capturing of the input signal. For proper

function the hold time has to fulfil the following relation to some of the other parameters ([6] eq. 8.3):

$$max(t_{hold}) < min(t_{flip-flop}) + min(t_{logic}) + min(t_{clock-skew(R-T)}) \quad \text{(J.2)}$$

After a clock transition, the fastest new signal will need a time $min(t_{flip-flop}) + min(t_{logic})$ to propagate downstream to the next latch. Since the signal at the latch input may change after this time, the hold time has to be shorter than indicated in the relation given above. The relation must also contain the minimum clock skew which may fall in the "disfavour" of the hold time, reducing the available maximum hold time further.

It is not uncommon to have a hold time which is zero or negative. Even with such hold times there may be a problem like when a flip-flop directly feeds another flip-flop and the clock skew is significant. In such cases dummy buffers have to be inserted to add delay, or the minimum flip-flop delay, $t_{flip-flop}$, has to be larger than the sum of the clock skew and hold time.

Race condition

While equation J.1 and equation J.2 are known from the literature we found that an equation was required to express the dependency of some latches on sharp clock edges.

$$min(t_{flip-flop}) + min(t_{logic}) + min(t_{set-up}) > max(t_{skew(\phi_R - \phi_T)}) + max(t_{skew:|\phi - \overline{\phi}|}) + \alpha \cdot max(t_{10\%-90\%}) \quad \text{(J.3)}$$

The inequality J.3 expresses the relation to be fulfilled to avoid signal propagation through more than one latch-pair interval. The left side is the minimum time a signal released by one clock needs to propagate to the next latch down-stream. The right side is, for the same clock edge, the time between the opening of one latch for data transfer through the clock barrier and the closing of the next latch down-stream. Clearly, the opening times can not be so long or so out of phase that data passing one latch can also reach to pass the next latch on the same clock edge. The components on the left side have been defined earlier. The term, $t_{skew(\phi_R - \phi_T)}$, is the time interval from a clock edge reaches latch T and its successor: R. The last two elements are used to express the raise/fall time of the clock edge. During rise/fall time a transistor or a latch is not completely switched off or on. Thus, if the raise or fall time is very long, data may propagate through a number of latches. Since a latch is not transparent during the entire raise/fall time it is multiplied by an efficiency factor α. The last term, $max(t_{10\%-90\%})$, is the raise/fall time at a latch of a clock signal with no clock skew. When both non-inverted and inverted clock signals are required $max(t_{skew:|\phi - \overline{\phi}|})$ is the clock skew between these clock signals.

J.2 Latches

Edge triggered clocking.

As indicated by the name, edge triggered flip-flops change value on the edges. This gives three types of flip-flops: Positive edge triggered, negative edge triggered and dual edge triggered. The dual edge triggered, reading data on both edges, have a data rate twice that of the single edge triggered.

Clock skew and clock frequencies.

As stated above high clock frequencies require minimum clock skew.

Factors that contribute to increased clock skew are
a) Different wire lengths,
b) Different number of buffers between the source and the latches using the clock,
c) Variations in clock buffer delays both on the same circuit and between clock buffers in different circuits,
d) Transmission line reflections on wires resulting in increased time before signals stabilise,
e) Wire resistance and capacitance contributes with a propagation delay and a signal outflattening, and
f) Different capacitive load.

• *Different wire lengths.* Variations in wire lengths may be reduced by routing according to a recursive H, X or tree structure. With these routing schemes latches may be placed at leaves of the structures with a similar distance to the source.

• *Different number of clock buffers between the source and the fed latches.* When buffer tree structures are used it is important to have an equal number of buffers between the source and the latches at the leaves of the clock tree.

• *Variations in clock buffer delays.* The variation in buffer delays may be 25 % ([19]) between two buffers on the same circuit. Between buffers on different circuits the variation may be even larger. With buffer delays from 400ps and upward, this may give a significant contribution to the clock skew. One way to avoid this is to have a common chain of increasing buffers, with a larger buffer at the end capable of driving all clocked latches. To reduce the clock skew further the clock signal should be distributed through a hierarchic H tree structure. In this way the latches will experience a clock signal where a clock edge arrives at latches almost simultaneously. One disadvantage is that the clock paths may be long, resulting in smoothed, delayed signals at the receiving ends. The well known equation $rcl^2/2$ ([109] eq. 4.18 p. 133) shows that the delay increases with the path length in the power of two. Thus, for long paths, shorter delays can be attained by dividing the path into several subsections with individual buffers between them. But then we have different buffers on the different paths with the delay variations we initially wanted to avoid. Another disadvantage of the long paths is that the clock signal will take a smoothed curve in the middle without reaching the rail values. This may result in that the clocked transistors are never turned completely on or off. Also the slower transition through the center section, may result in that latches trigger at different times resulting in increased clock skew.

J.3 Single clock latches.

Figure J.1 and J.2 show two latches fed by a single clock. Some of their characteristics are shown in table J.1.

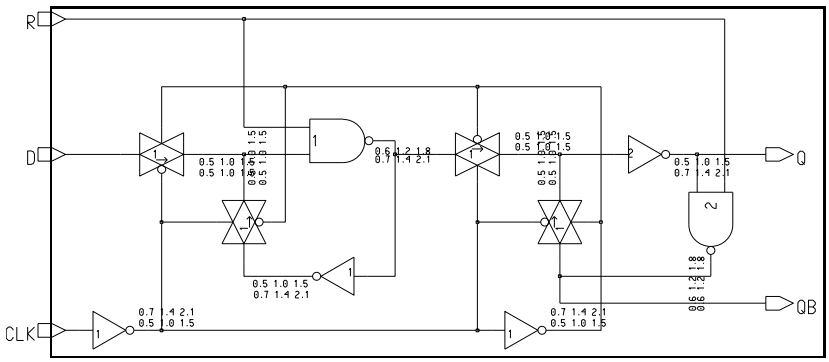


Figure J.1: Static latch. Local clock signal and local inverted clock signal are generated from the global clock signal through two inverters. Thus the latch is less sensitive to the raise and fall time of the fed clock signal. The latch has 24 transistors.

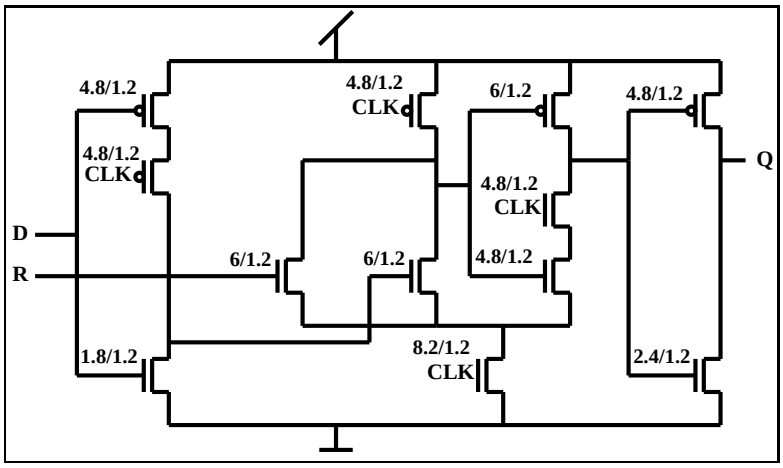


Figure J.2: Dynamic latch. Correct operation requires that the raise and fall times are not too long. The latch has 12 transistors.

The advantages of the dynamic latch compared to the static latch.

The dynamic latch allows a clock period $1.45ns$ shorter than that of the static latch. Thus the dynamic latch may be regarded as better suited for the highest frequencies. It also has the advantage of being smaller in area and number of transistors.

The disadvantages of the dynamic latch compared to the static latch.

Both analog and digital simulators can simulate the static latch while some digital simulators will have a problem with the dynamic latch. Another problem with the dynamic latch is that it is depending on sharp clock edges. If the edges are too slow, data may propagate through several latches on one clock edge.[9] Thus, if for some reason correct circuit behaviour should be

[9]For two successive latches as in figure J.2 simulations show signal race for rise and fall times above $9ns$. This corresponds to $\alpha = 1/5$ in equation J.3. For scaled latches with different transistor lengths simulations show a critical rise/fall time of $12.5ns$ for $L = 1.6\mu m$, $5.0ns$ for $L = 0.8\mu m$ and $3.0ns$ for $L = 0.6\mu m$.

	Latch in figure J.1	Latch in figure J.2
Externally fed clock	Single phase	Single phase
Latch type	Static	Dynamic
No. of transistors	24	12
$t_{flip-flop}$	$2.3ns$	$0.55ns$
t_{setup}	$1.2ns$	$1.5ns$
t_{hold}	$1.0ns$	$-0.05ns$

Table J.1: Characteristics and simulated values for two latches fed by a single clock. The simulated values are based on a typical $1.2\mu m$ *CMOS process.*

attained through reduced clock frequencies, the clock edges still have to be sharp to guarantee correct latch functions. This is often a problem in dysfunctional circuits since slow clock edges often are a part of the problem. In the static latch, the clock signals are refreshed locally and sharper clock-edges are achieved. Another advantage of the static latch is that correct behaviour may also be achieved when the fed clock does not go all the way to the rails[10]. The inverter at the clock input will generate proper internal clock signals. This will not be the case for the dynamic latch drawn, for which reduced clock swing will result in that transistors may never be turned off.

[10] Due to RC -effects in the routing wires.

Fast inverter[1]	0.4ns	
Inverter	0.6ns	
	NAND	NOR
2-input	0.8ns	1.0ns
3-input	1.1ns	1.7ns
4-input	1.3ns	2.2ns
5-input	1.7ns	3.2ns
6-input	2.0ns	3.9ns
7-input	2.5ns	
8-input	2.8ns	

Table J.2: Example of simulated delays for a 1.6μm CMOS process. The delays have been found by simulating chains consisting of ten equal gates. The gates have minimum n-transistors and balanced p-transistors according to the mobility relation between the two types of transistors.

[1]: The fast inverter has two parallel n-transistors and two parallel p-transistors.

	Minimum		Typical	Maximum	
		Rel. to typ.			Rel. to typ.
2-input NAND	0.54ns	-5%	0.57ns	1.0ns	+75%
3-input NAND	0.81ns	-9%	0.89ns	1.6ns	+80%
4-input NAND	1.1ns	-8%	1.2ns	2.2ns	+83%
2-input NOR	0.65ns	-19%	0.8ns	1.45ns	+82%
3-input NOR	0.72ns	-20%	0.9ns	1.5ns	+67%
inverter	0.32ns	-14%	0.37ns	0.65ns	+76%

Table J.3: Example of simulated variations in delays between different circuits processed with the same 1.2μm CMOS process. The variation inside each circuit is considerably less.

J.3.1 Logic gate delays.

Table J.2 and J.3 have been included to illustrate typical gate delays. Table J.3 shows the delay times for minimum, typical and maximum parameters for a 1.2μm process. Simulation of 15 different standard gates show a delay variation between -20% and $+80\%$ of the typical values.

The variations inside each circuit are less. Numbers from a commercial process house indicate worst-case variations up to 7.5% for the smallest transistors with worst orientation and large distance. Except for layers dedicated for resistances and capacitances, no further on-chip variations was given from the process house. For the dedicated resistance layer relative variations up to 0.25% was found for the smallest allowed sizes. For the capacitance layers the relative variation could be above 1.0% for small sizes.[11] If routing metal is processed with the same precision, variations between two metal pieces with minimum width and 100μm length would be less than

[11] For more "normal" sizes where matched elements are placed close and layout strategies to reduce mismatch have been used, a relative variation in resistance and capacitance less than 0.1% may be achieved.

1.5%. In [19] (page 635) on-chip gate delay variations around 20% were estimated. Variations in fabrication, supply voltage and temperature were included. Obviously the number is depending on the chip size and the design strategy. Thus the number should be regarded only as an indication of in what area the numbers will be.

The conclusion from these numbers is that between different circuits a variation range around the typical gate delay may be -20 % to +80 %. This gives an indication of local signal and clock skew when a global signal is buffered locally in several circuits. We also see that for a circuit to operate also with the maximum process parameters, the signal should use half of the maximum time when simulated with typical parameters. This is if we want to accept all circuits declared "good" by the foundry.

J.4 Wire propagation time and capacitive coupling.

Simulation of standard digital circuitry is simpler with low clock rates than with high clock rates. At low clock rates correct behaviour may be verified through simulation on a net list of transistors or library cells. Thus no information of the circuitry layout is required. As the clock period decreases or wire lengths increase, signal propagation time will constitute an increasing part of the clock period. When this happens, the following actions may be taken:
- Wire routing information has to be included in the simulations.
- New layout design rules have to be added:
  - — To reduce the signal propagation time.
  - — To make it more predictable.

We will in the following get an impression of the size of the signal propagation time. Some calculations required for a layout with more control of the signal propagation time is presented.

Also very important is the transfer of potential changes between wires due to the capacitive coupling between wires.

J.4.1 Routing capacitance.

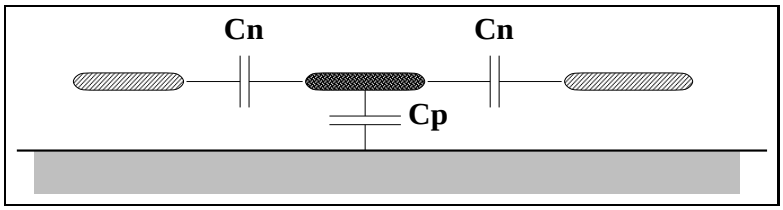


Figure J.3: Capacitances experienced by a signal line. End capacitances are not included. Neither are capacitances from crossing lines. C_n is the capacitance to a parallel line of same material and level. C_p is the capacitance to a plane below (and optionally above) with constant potential level.

The capacitive couplings between a metal 2 or metal 1 wire and its surroundings are as follows:
- A capacitive coupling to other wires on the same level. This is typically parallel and thus significant.
- A capacitive coupling to the other metal layer. This may be ignored. According to normal lay-out strategies this routing would be orthogonal and thus the capacitive coupling to each line

would be small. Also the percent of routing which is crossed by wires from the other metal layer is small.
• Capacitive coupling to a "plane" below (or above). In low clock rate design this is the substrate. In designs for higher clock rates, a poly or metal plane may be included for reasons discussed later in this appendix.

The first and third capacitances above are illustrated in figure J.3.

The signal propagation time for a model where the wire consists of small segments with resistance r and capacitance c, may be expressed by the following equation from ([109] eq. 4.18 p. 133):

$$t_{wp} = rcl^2/2 \tag{J.4}$$

Here, t_{wp} is the signal propagation time along the wire. We express the resistance, r, pr. μm as:

$$r = R/(w \cdot w_0) \tag{J.5}$$

where R is the resistance in $\Omega/\Box$, w_0 is the minimum width of the wire according to the design rules and w is the wire width in w_0. We express the capacitance c pr μm as:

$$C_{p0} \cdot w \cdot w_0 + 2 \cdot C_{n0}/s \tag{J.6}$$

C_{p0}[12] is the capacitance per area to a plane below and possibly also above, with fixed power. This may be the substrate, a poly layer or a metal layer. C_n[13] is the capacitance to a parallel neighbour line with minimum space according to the design rules. The lowercase s is the space between the parallel wires expressed as a multiple of the minimum width. By putting these values into eq. J.4 we achieve:

$$t_{wp} = \frac{rcl^2}{2} = \frac{\frac{R}{w \cdot w_0}(C_{p0} \cdot w \cdot w_0 + 2\frac{C_{n0}}{s}) \cdot l^2}{2} = \frac{R \cdot C_{p0}}{2} l^2 + \frac{R \cdot C_{n0}}{w_0} \frac{l^2}{w \cdot s} \tag{J.7}$$

The element $R \cdot C_{p0} \cdot l^2/2$ is the part of the wire propagation time due to capacitance to the power plane. We denote this element t_p in table J.4. The element $(R \cdot C_{n0} \cdot l^2)/(w_0 \cdot w \cdot s)$ is the contribution due to the capacitance to neighbour parallel lines. We denote this element t_n. In table J.4, with $s = 0$ and $w = 0$, t_n is denoted t_{n0}. t_n may give a significant contribution and we may want to reduce it by increasing w and s.

Relation between signal propagation time and raise and fall time.

If the signal propagation time is short compared to the raise and fall time, the line may be regarded as a distributed capacitance. Voltage and current waves back and forth between non-homogenous points have time to stabilise as the potential increases or decreases. According to [6] eq. 6.31 a line may be considered as a capacitance when the allowed raise and fall times are larger than five times the signal propagation time. This is typical at low clock rates or for small circuits.

When the signal propagation time increases or the raise and fall times decrease, potential and current waves have not time to stabilise and transmission line phenomena occur. This complicates the circuit layout and is avoided whenever possible. Special cautions have to be taken during

[12] "p" for power
[13] "n" for neighbour.

Wire length:		10mm		20mm		30mm		40mm		
Example chip size		($25mm^2$)		($100mm^2$)		($225mm^2$)		($400mm^2$)		
		t_p	t_{n0}	t_p	t_{n0}	t_p	t_{n0}	t_p	t_{n0}	
metal 2 wire	min.	0.04ns	0.09ns	0.17ns	0.34ns	0.38ns	0.77ns	0.67ns	1.37ns	-12%
above	typ.	0.05ns	0.09ns	0.19ns	0.34ns	0.43ns	0.77ns	0.77ns	1.37ns	
substrate	max.	0.06ns	0.09ns	0.23ns	0.34ns	0.51ns	0.77ns	0.91ns	1.37ns	+ 19%
metal2 wire	min.	0.05ns	0.09ns	0.22ns	0.34ns	0.49ns	0.77ns	0.86ns	1.37ns	- 15%
above	**typ.**	**0.06ns**	**0.08ns**	**0.25ns**	**0.34ns**	**0.57ns**	**0.77ns**	**1.01ns**	**1.37ns**	
poly1	max.	0.08ns	0.09ns	0.31ns	0.34ns	0.70ns	0.77ns	1.25ns	1.37ns	+ 24 %
metal2 wire	min.	0.08ns	0.09ns	0.34ns	0.34ns	0.76ns	0.77ns	1.34ns	1.37ns	-20%
above	typ.	0.11ns	0.09ns	0.42ns	0.34ns	0.95ns	0.77ns	1.68ns	1.37ns	
metal1	max.	0.14ns	0.09ns	0.55ns	0.34ns	1.24ns	0.77ns	2.21ns	1.37ns	+ 32 %
metal1 wire	min.	0.11ns	0.17ns	0.43ns	0.67ns	0.97ns	1.50ns	1.73ns	2.67ns	-42%
above	typ.	0.12ns	0.17ns	0.46ns	0.67ns	1.04ns	1.50ns	1.86ns	2.67ns	
substrate	max.	0.12ns	0.17ns	0.50ns	0.67ns	1.12ns	1.50ns	1.98ns	2.67ns	+7 %
metal1 wire	min.	0.20ns	0.17ns	0.78ns	0.67ns	1.76ns	1.50ns	3.14ns	2.67ns	-8 %
above	**typ.**	**0.21ns**	**0.17ns**	**0.85ns**	**0.67ns**	**1.91ns**	**1.50ns**	**3.39ns**	**2.67ns**	
poly1	max.	0.23ns	0.17ns	0.93ns	0.67ns	2.09ns	1.50ns	3.71ns	2.67ns	+10%

Table J.4: Simulated metal 1 and metal 2 propagation time elements on an integrated circuit. The values are for a typical $1.2\mu m$ CMOS process. The table contains entries for metal 1 and metal 2 wires above several different power planes. The neighbour-dependent part of the propagation time, t_{n0} is given with $s = 1$ and $w = 1$. A doubling of the metal width and the space between the wires ($s = 2$ and $w = 2$) will reduce t_n to one fourth. The example chips are regarded as square and the wire length equal to two edges. The variations given in the last column is for t_p alone.

circuit layout at the line driver, at the line reader and for the line routing. In [6] (eq. 6.32) transmission line phenomena are expected to be significant when the raise and fall times are less than 2.5 times the signal propagation time.

With a raise and fall time of one ns, the wire is regarded as a capacitance when the signal propagation time is less than 0.2ns. This is satisfied for metal 2 wires above poly 1 or substrate with a length less than 20mm or for metal 1 wires with a length less than 10mm. With the same raise and fall time, the signal line has to be regarded as a transmission line if the signal propagation time is above 0.4ns. The propagation time is above 0.4ns for all metal 2 wires with a length above 30mm and all metal 1 wires with a length above 20mm.

As the clock periods decrease or the circuit size increases, layout of wires get more complicated. This is a difficult topic requiring knowledge and experience. The topic is important when more circuitry is to be included on one circuit or when the clock rates increase to support higher bit rate and lower data propagation time. However, due to the many other aspects of multicomputer systems which require analysis, it has been decided to postpone advanced clocking design to later switches. Thus the present SWIPP circuits have not been designed with the highest clock rates the technology can support. To fulfil the requirement that the lines should be regarded as capacitances only, one ns raise and fall time is a minimum, with the circuit sizes we want to use. With a raise and fall time of one ns a clock periode of five to ten ns is suitable.

J.4.2 Capacitive coupling.

An important effect to be considered for higher clock rates is the capacitive coupling between neighbour lines. A potential step on one line will initiate a potential step on neighbour lines. The magnitude of the initiated potential step depends on the relation between capacitances as expressed in the following equation (based on [6] p. 11 eq. 1.4).

$$\Delta V_{\text{Generated in target}} = \Delta V_{\text{Source}} \frac{C_{\text{ Target- Source}}}{C_{\text{Target-Source}} + C_{\text{Target-Power plane}} + C_{\text{Target-Other neighbours}}} \quad \text{(J.8)}$$

The noise generated from a neighbour wire is a function of the capacitance between the target and the noise source divided by the total capacitance experienced by the target line. By using the parameters from a typical $1.2\mu m$ CMOS process we find that with metal 1, with minimum line width, minimum space between metal lines and worst-case process parameters but with a good conducting power plane, 50% of the potential change on the source line is transferred to the target line. If the power plane is a bad conductor (like the field substrate) the situation will be worse. If a line has two other lines on each side, 33 % of the potential change from each of them is transferred to the one in the middle. The capacitive coupling may be reduced by increasing the width of each line (w) and increasing the space between lines (s). It may also be reduced by having an increased capacitance to a stable potential like a metal layer or poly layer above or below. This last possibility will, on the other hand, increase the signal propagation time.

If the line width is increased to twice the minimum size, and the space to twice the minimum space, the potential transfer from one neighbour line would be 20 %. If a line has one line on each side the potential transfer would be 17% from each.

The generated potential step initiates potential waves in the target line which have to be compensated by the circuits driving the target line. This compensation has to be large enough and

fast enough to allow data to be read correctly at the next clock edge.

J.5 Clock routing.

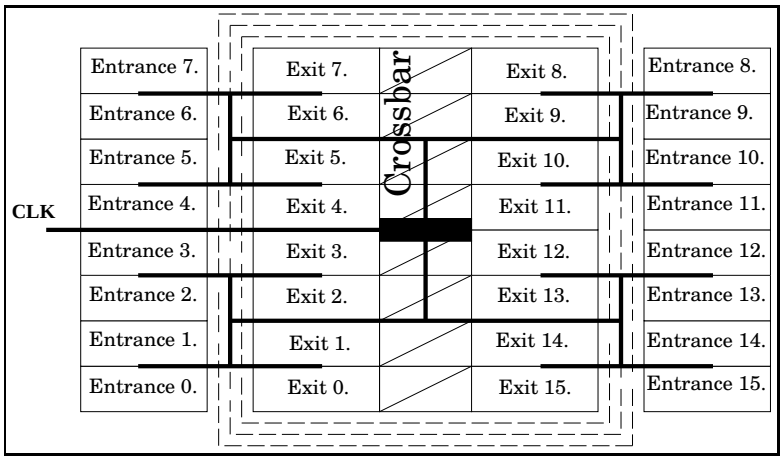


Figure J.4: A possible routing pattern for the clock signal on the CSU. The "H"-structure will result in that the length of the clock path is similar for all receivers. With a chip size of $100mm^2$ *square the length of the clock path is below* $20mm$*. If the clock signal is routed in metal 2 above substrate the propagation time is* $0.19ns$ *while it would be* $0.46ns$ *if routed in metal 1. Thus, to reduce the propagation time it is important that as much as possible of the routing is done in metal 2. To secure simultaneous arrival of the clock signal to all destinations it is important that the part routed in metal 2 and metal 1 are equal for all clock branches. If other signals are routed in the neighbourhood of the clock signal, the conduction in the power layer below has to be strengthened to reduce the capacitive coupling. A solution is to have a poly 1 layer below all clock lines, connected to one of the power potentials. The disadvantage is that this would increase the propagation time to* $0.25ns$ *for metal 2 and* $0.85ns$ *for metal 1. Thus the need to route the clock signal entirely in metal 2 is increased. The clock line should be wide to reduce both the resistance and the capacitive coupling.*

Appendix K: Implementation of the switch board

The final design goal for SWIPP is to implement the entire switch as one integrated circuit. With the present layouts and their technologies this is not possible. Redesign of layout for the newest technologies (table 4.2) should make it possible to implement the CSU together with Input Ports, Output Ports, some memory and the circuitry for conversion between parallel and serial representations (for fibre or coaxial cable transmission). This appendix presents a proposal for how the switch circuits may be put together with the present layouts in their technologies. This appendix focus on how the circuits may be put together, choice of substrate, signal routing between circuits and heat dissipation. Some of the trade-offs are also discussed. This appendix contains a mixture of textbook material and the author's own results and estimates.

Figure K.1 shows a layout proposal for the switch board. In the proposal the switch is built of 17 custom designed integrated circuits on a multilayer substrate. Some discrete components for connection of fibres are not drawn. The substrate is expected to have a size of 6 $cm \times 8$ cm. The CSU and 2IP/OP circuits are designed in CMOS while the 2OMT/OMR circuits require higher clock rate and should be implemented in BiCMOS.

The main design challenges are the requirement for high channel bandwidth and low power consumption, and the conflict between these two design goals. The switches are intended not to have fans, so all heat transport has to take place through natural convection and conduction. More specific design challenges are:
• High number of pins on some of the circuits.
• Low capacitance on inter-chip wiring. This is to reduce the power consumption due to capacitive load. It is also to avoid transmission line behaviour on inter-chip lines. This is both to support higher clock rates and to avoid the power consuming termination resistors required for transmission lines.
• Short (low inductance) power lines. This is to reduce voltage variations on power lines due to current spikes. Variations in supply potentials may result in errounous interpretation of local potential. The consequence may be wrong logical levels.
• Little disturbance from capacitive coupling. This requires that the dominating capacitances experienced by a signal line should be to a strong, fixed potential.
• Good heat conduction between circuits and substrate.

These design challenges influence our choice of substrate materials, the wiring between circuits and how the circuits are mounted.

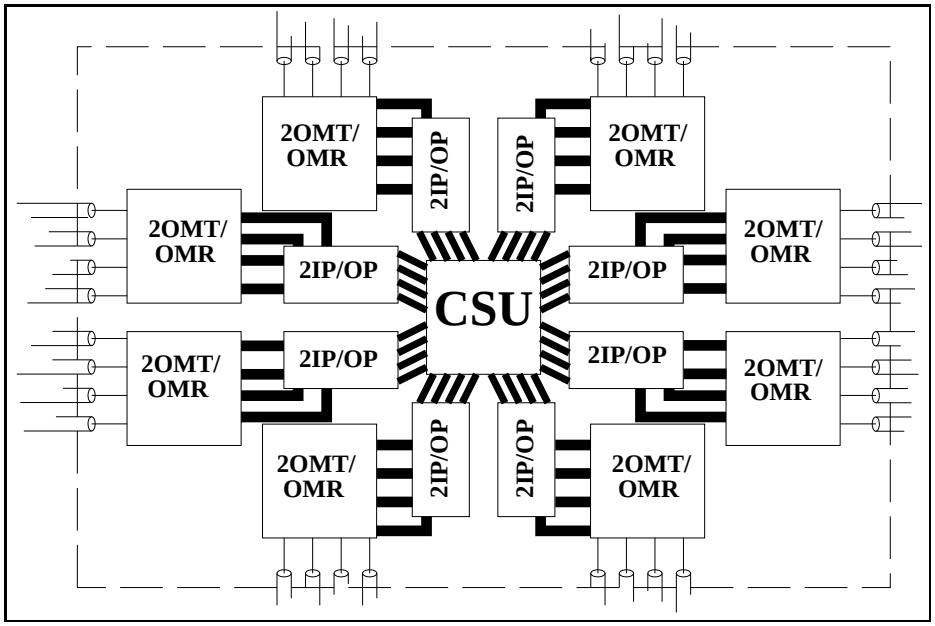


Figure K.1: Proposal for a complete switch board with full custom integrated circuits. The board consists of the CSU circuit, eight 2IP/OP circuits and eight 2OMT/OMR circuits. Each of the 2IP/OP circuits contains two Input Ports and two Output Ports and two blocks of memory, one for each Input Port. Each 2OMT/OMR circuit contains two encoders and two decoders for conversion between our parallel and serial bit formats. The positions of the circuits are chosen to give minimum wire lengths between the circuits.

K.1 The data buses.

The data buses on the circuit board have a bandwidth of either 900 Mbit/s or 1000 Mbit/s. These buses are implemented in three combinations of width and line bit rate: 5-line-wide buses with a line bandwidth of 200 Mbit/s (between the CSU and the 2IP/OP circuits), 9-line-wide buses with a line bandwidth of 100 Mbit/s (between 2IP/OP circuits and the Optical Modules), and a single line with a bandwidth of 1000 Mbit/s (between the Optical Module and the fibre). The buses have one transmitting and one receiving end.

K.2 The CSU circuit.

Pad number and pad positions.

The CSU requires pads for data buses, clock signal, reset signal and power. On a substrate with only one signal layer the data buses, clock signal and reset signal have to be routed on this signal level. The pitch of the wires on the substrate[14] has to be at least as small as the pitch of the

[14] Here the pitch is one line width and one line space.

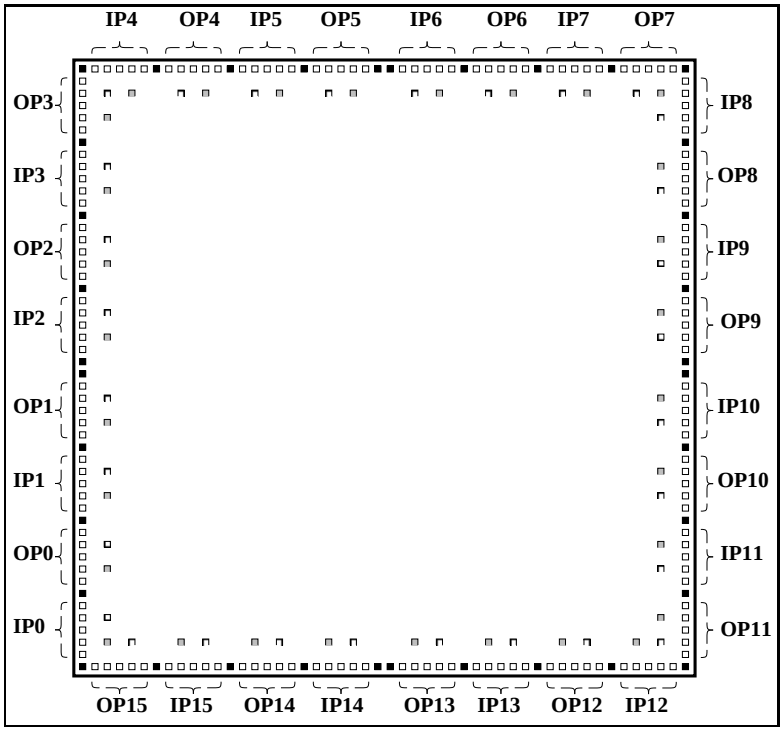


Figure K.2: Bonding pad proposal for the Central Switch Unit. The pads for data buses, clock signals and reset signals are in one row close to the circuit edges. The power pads are placed inside this outer row. The figure shows only a few power pads. More pads should be added to increase the thermic and electrical conduction between the circuit and the substrate.

circuit pads[15]. A standard pad offered by several process houses has a size of $100\mu m \times 100\mu m$ and a pitch of $200\mu m$. Pads with smaller pitch are available. With multiple pad rows, more than two pads may be connected for each $100\mu m$ of chip edge. For wiring on substrate a pitch of $200\mu m$ is small. The smallest standard wire width is $150\mu m$(6 mil) giving a wire pitch of $300\mu m$.

We have chosen a wire and pad pitch of $200\mu m$ which is conservative for the circuits but advanced for the substrate. Thus all signal pads (data buses, clock and reset signals) can be placed in one row at the circuit edges. With 16 incoming and 16 outgoing data buses, each with a width of 5 lines, $32mm$ of edge length is needed for the 160 wires for data buses. With a few additional mm for clock and reset signals the circuit size decided by the pads is close to $10mm \times 10mm$.

K.2.1 Power consumption in the CSU.

Estimate of internal power consumption.

The internal dynamic power consumption can be expressed with the following equation from chapter 20:

$$P = \frac{1}{2} \cdot N_t \cdot f_d \cdot f_{clk} \cdot Vdd^2 \cdot C_{Load} \tag{K.1}$$

[15]Here the pitch is one pad and one space between pads.

where N_t is the number of transistors, f_{clk} the clock frequency, Vdd the power voltage and C_{Load} the capacitive load of each gate. The factor f_d represents the proportion of transistors changing state. The arguments for choosing f_d equal to 1/4 was given in chapter 20. We estimate the average number of transistors to approximately 60000. f_c is the internal clock rate which is 100 MHz. A significant part of the logic (more than a half) is only changing at packet start and packet end. This part of the logic will efficiently be clocked at a frequency of 100 MHz divided by the packet length in bytes. Some logic at the bus inputs and outputs is clocked at 200 MHz.

The capacitive load per node is estimated to $(x/1.2\mu m)^2 \cdot 100fF$ where x is the minimum gate length of the technology. These numbers give an estimated power consumption of 1.9W with the 5V power supply. Table 4.3 shows how the power consumption can be reduced by reducing the power supply while the same bandwidth is maintained through increased parallel width. In this case the 1.9W at 5V should give 1.3W at 3.3V, 1.1W at 2.5V, and 0.75W with a 1.5V power supply. This reduction in power consumption is based on the same technology. Smaller transistor sizes would give a further reduction in power consumption due to the decreased capacitive loads.

K.3 The 2IP/OP circuits.

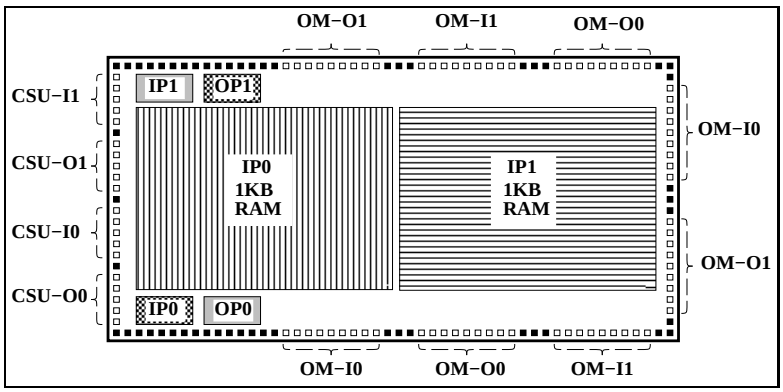


Figure K.3: The 2IP/OP circuit with double set of 9-bit buses.

The Input/Output Port circuits have a size of $5mm \times 10\ mm$. Two 1kbyte memories with a size of $4mm \times 4mm$ each, cover the broader part of the circuit. Each circuit contains two pairs of Input and Output Ports, one pair for each channel. The 2IP/OP circuits have two five-line-wide input buses and two five-line-wide output buses for connection between the CSU and the IP/OP circuit. The IP/OP circuits also have two 9-line-wide input buses and two 9-line-wide output buses for connection to the Optical Modules. Each 9-line-wide bus has two alternative connection points to the IP/OP circuits. The circuits are mounted as flip-chip (with the surface downwards).

K.3.1 Power consumption in the Input Port.

The dominating part of the Input Port is the Input Buffer. As stated in *18 Implementation of the elastic FIFO* the Input Port may be implemented as a one-port static RAM to save space. The RAM may operate at a lower clock rate than the remaining part of the switch. Only one word at a time is active in the RAM. This gives a lower power consumption per transistor compared to the remaining parts of the switch.

The number of transistors in the high-clock-rate part of the Input Port is approximately 4000. With a similar calculation as for the CSU, the power consumption in this part is $125mW$ with a $5V$ supply and a clock rate of 100MHz. With a power of $3.3V$ the power consumption is expected to be $55mW$. With $2.5V$ it would be $32mW$, and with $1.5V$ it would be $12mW$.

K.3.2 Power consumption in the Output Port.

The number of active transistors in the Output Port is approximately 2000. With a similar calculation of the static power consumption to that performed for the CSU and Input Port the power consumption is $65mW$ with a $5V$ supply, $27mW$ with a $3.3V$ supply, $17mW$ with a $2.5V$ supply and $6mW$ with a $1.5V$ supply.

K.4 The Optical Modules.

Only the low-speed digital part of the Optical Modules has been designed. Thus size can only be roughly estimated. A circuit of $10mm \times 10mm$ is expected to contain two transmitter and two receiver blocks, one for each channel. The receiver blocks contain both digital encoding logic and an analog phase-locked loop. The low-speed parts have been designed in CMOS while the high-speed parts require bipolar transistors. Thus a high-speed BiCMOS technology would be the best solution.

Each OM circuit has two incoming and two outgoing 9-line-wide buses. Data from the two incoming buses are transmitted on two high-data-rate serial lines to light sources. Data from light detectors received on two high-data-rate serial lines are transmitted to the two outgoing 9-line-wide buses.

For the first prototypes a temporary solution may be to use standard components for the first switch versions.

K.4.1 Power consumption in the Optical Module.

Only the 100MHz part of the Optical Module has been designed so far. Due to several "carry"[16]-like signals up and down the word widths, it is not easy to reduce power consumption further through reduced power voltage and increased parallelity.

The part of the circuit operating on one GHz has not yet been designed. This part clearly will be power consuming and it is difficult to find methods to reduce the power consumption.

Due to some holes in the knowledge about implementation of this part we may use a similar, commercially available module as a referance. In [80] a similar transmitter and receiver, both operating on $5V$, is presented. The power consumption of the transmitter is $2.0W$ while the power consumption of the receiver is $2.6W$.

[16] I.e. carry in addition / subtraction circuits.

K.5 Substrate.

The substrate consists of multiple layers of conducting and isolating material. The signal wiring is on top of the substrate, closest to the circuits. The substrate has several functions:
• The isolating and conducting elements of the substrate are electrical components influencing on the electrical behaviour of the total system.
• The substrate is a heat conductor carrying heat away from the circuits.
• The substrate is a mechanical basis for circuits and wiring. Its combination of elasticity and temperature expansion factors results in that connections are kept intact within given spesificatins.

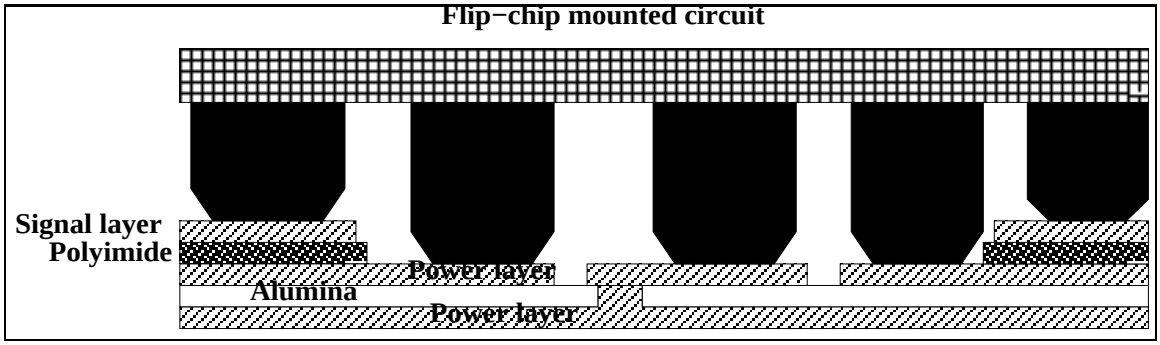


Figure K.4: Flip-chip mounted circuit on substrate layers as described in this appendix. The signal layer on top of the substrate contains wiring for data buses, clock and reset signals. The signal layer is isolated from the remaining part of the substrate by a low dielectric Polyimide layer. The conducting layers with the power potential is isolated from each other by an Alumina isolator having a high dielectric constant. The signal pads are at the edge of the circuit while the power pads are inside the outher circle. For better electrical and thermic conduction the power bumps go directly to the upper power layer. Typical bump heights are $60\mu m$ *and* $80\mu m$*. The bumps can adapt differences of at least* $10\mu m$*.*

The power layers and the isolation layers between them.

To get low electrical and thermic resistance each power potential covers entire layers of conducting material. The power levels are separated by an isolating material with a high thermic conductivity, a thermic expansion factor close to the silicon circuits, and a high dielectric constant. Preferred material to separate power layers are Silicon Carbide (SiC), with Alumina (Al_2O_3) as a second choice. The parameters for some common substrate materials are given in table K.1. They are preferred due to high thermic conductivity and high dielectric constants. Their thermic expansion factors are not far from the expansion factor of the integrated circuits. The high dielectric constant gives high decoupling capacitors resulting in reduced noise on power lines ($C = (A/t)\epsilon_r\epsilon_0$). Good decoupling close to the circuits is important for attaining low error rate on the data signals.

The signal layer and the isolation towards the remaining substrate core.

The inter-chip signal lines require substrate surroundings with a low dielectric constant. Low dielectric constant results in increased signal propagation speed and low wire capacitance. Higher signal propagation speed increases the wire length (alternatively increases the clock rate) before transmission line effects become significant. The propagation dependency on the dielectric constant may be expressed by $v = c/\sqrt{\epsilon_r}$ ([6] eq. 6.12). If the line has to be considered as a transmission line (discussed further later in this appendix), the wiring may have to be designed for a spesific impedance. The receiving end has to be terminated with a resistance equal to the characteristic impedance. For large voltage swings the power consumption through the termination resistance may be significant. A voltage swing of $5V$ through a 50Ω resistance would give a power consumption of $0.5W$ while a voltage swing of $0.8V$ through a 110Ω termination resistance would result in $6mW$. When the line does not have to be regarded as a transmission line, it is terminated by the high resistance input of the receiving transistors. Polyimide, with a relative dielectric constant in the range $2.5 - 3.5$, gives a signal propagation speed of $19cm/ns$ ([6] p 329) while alumina, with a dielectric constant of 9.5, gives a signal speed of $11cm/ns$. Further discussion of wire lengths, propagation speed, clock rate and transmission characteristics is given in 22 of this appendix. For short wire lengths the wires may be regarded as capacitances only. For a wire above a thin polymid isolator ($2\mu m$) with a wire width of $100\mu m$, the capacitance is $15pF/cm$. Above an alumina isolator of the same thickness the capacitance would be $43pF$ per cm of wire length. When calculated for an alumina isolator a voltage swing of $5V$ at $200MHz$ would give $107.5mW$ per cm. A voltage swing of $0.8V$ would give $2.75mW$. The same calculations for a polyimide isolator would give $37.5mW$ for a $5V$ swing and $0.96mW$ for a $0.8V$ swing. When multiplied with the number of wires on the switch board the power consumption from bus driving becomes very significant. At the board design level, these numbers emphasise the need to make connections between circuits as short as possible.

One way to reduce the capacitive load (and the power consumption) of the wires is to increase the thickness of the isolation layer. The capacitive load is inversely proportional to the thickness. The maximum thickness of the isolation layer is restricted by the capacitive coupling effect. The consequence of this effect is that more than half of the capacitance of a conductor has to be directed towards a conductor with a stable potential. With a pad pitch of $200\mu m$ on integrated circuits the wires have to have the same pitch. Normally the wire width and wire space are equal. To make the capacitance to the power level below the dominating one, the absolute maximum thickness is $50\mu m$ with a preferred maximum between $40\mu m$ and $30\mu m$.

Copper has a low electrical resistivity ($\rho = 1.7\mu\Omega - cm$) making it suitable as a wiring material for the bus connections on top of the substrate. Molybdenum or Tungsten may be better choices than Copper for the power metal layers inside the substrate, due to melting points better suited for processing with the chosen isolation material.

K.6 Flip-chip mounting.

The meaning of "flip-chip" mounting is that a naked circuit is placed directly on the substrate with the component side of the circuit directed towards the substrate. The circuit is connected electrically, mechanically and thermically to the substrate through soldering bumps. To increase the thermic conduction an electrically isolating but thermically conducting material may be put between the circuit and the substrate ("underfill").

Material	Thermal Conductivity $W/cm \cdot {}^{O}K$	Their Coefficiant of Thermal Expansion $10^{-6} \cdot 1/{}^{O}K$	Dielectric Constant
Metal			
Copper	4.0	17.0	
Tungsten	1.7	5.0	
Molybdenum	1.4	5.0	
Semiconductor			
Silicon	1.5	2.5	11.8
Insulating substrates			
Silicon Carbide (SiC)	2.2	3.7	42.0
Alumina (Al_2O_3)	0.3	6.0	9.5
Polyimide	0.004		3.5

Table K.1: Parameters for some common metal, semiconductor and isolating materials used in substrates.

Component	Capacitance (pF)	Inductance (nH)
68-pin plastic DIP (without ground plane)	4	35
68-pin ceramic DIP pin (with ground plane)	7	20
68-pin SMT chip carrier lead (without ground plane)	2	7
68-pin PGA pin (without ground plane)	5	15
Wire bond	1	1
Solder bump	0.5	0.1

Table K.2: Capacitances and Inductances of Various Package Components.

Compared to other methods flip-chip mounting has the following advantages:
• The position of pads is not limited to the circuit edges. This both allows a freer placement and a higher number of pads.
• Since the "connection wires" are below the circuit, less space is required for each circuit and they may be placed more densely.
• The wiring length between circuits may be made shorter.
• The "connection wire" between the circuit and the substrate has a lower capacitance (Table K.2). This reduces the power consumption on signal lines. It may also be used to allow sharper raise and fall times driven by CMOS circuits.
• The smaller wire inductance on the power connections is important. The smaller inductance together with the decoupling capacitance of the power plane reduces voltage variations due to current spikes.

Table K.2 shows that flip-chip mounting gives approximately half of the capacitance and a tenth of the impedance of a wire-bonded circuit. Compared to packed circuits the advantage is larger.

K.6.1 The disadvantages of flip-chip mounting of circuits.

Flip-chip mounting gives a less elastic "wire" connection between circuit and substrate than other circuit mounting techniques. Differences in temperature or temperature expansion factors give mechanical stress in bonding points. This stress may result in broken connections.

Methods to reduce mechanical stress due to temperature.

This stress is reduced by one or more of the following methods:
- Equal temperature expansion factors on circuit and substrate, and
- Placing bonding pads close to the circuit center.
- Temperature reduction through:
  —Adding additional bonding pads (preferably for power),
  —Using "underfill" i.e. a thermically conducting but electrically isolating material between the circuit and the substrate.

We choose all except the second alternative.

K.7 Inter-chip transmission line analysis.

If the raise and fall times are much shorter than the time the signal needs to propagate along the line, the line must be considered as a transmission line with an impedance. As a rule of thumb, transmission line phenomena become significant ([6] eq. 6.31), when:

$$t_r, t_f < 2.5 t_p \tag{K.2}$$

t_r is the rise time, t_f the fall time, while t_p is the signal propagation time along the line from transmitter to receiver. t_r and t_f are expected to be similar so only t_r will be used in the following discussion. By substituting t_p with $l\sqrt{\epsilon_r}/c$ we can get the following lower limit for l:

$$l > \frac{t_r \cdot c}{2.5 \cdot \sqrt{\epsilon_r}} \tag{K.3}$$

If the raise and fall times are much longer than the signal propagation time, the line may be considered as a lumped capacitance. As a rule of thumb ([6] eq. 6.32) this occurs when:

$$t_r > 5 \cdot t_p \tag{K.4}$$

By substituting in a similar way as for the previous case, we get:

$$l < \frac{t_r \cdot c}{5 \cdot \sqrt{\epsilon_r}} \tag{K.5}$$

The lengths in table K.3 are found with a dielectric constant for polyimide of $\epsilon_r = 2.5$. A dielectric constant of $\epsilon_r = 10$ would give lengths half of the values in table K.3 (and four times as large capacitance).

Typical off-chip rise/fall times for CMOS range in the area 2 - 4 ns (Tab. K.4). From table K.3 we find that for our small substrate the 100 Mbps and the 200 Mbps buses may be treated as

$t_r(\approx t_f)$	Transmission line when larger than:	Capacitance when smaller than:	Raise/fall time part of data periode		
			20 %	50 %	100 %
0.1 ns	0.76 cm	0.38 cm			
0.2 ns	1.5 cm	0.76 cm	1 Gbps		
0.5 ns	3.8 cm	1.9 cm		1 Gbps	
1.0 ns	7.6 cm	3.8 cm	200 Mbps		1 Gbps
2.0 ns	15 cm	7.6 cm	100 Mbps		
2.5 ns	19 cm	9.5 cm		200 Mbps	
5.0 ns	38 cm	19 cm		100 Mbps	200 Mbps
10 ns	76 cm	38 cm			100 Mbps

Table K.3: Transmission line behaviour as a function of length, calculated for a isolation layer of polyimide with a di-electric factor $\epsilon_r = 2.5$. The conclusion is that, so far, the clock rates are low enough and the wires can be made short enough that all wires may be regarded as pure capacitive load.

Technology	On-chip rise/fall times	Off-chip rise/fall times
CMOS	0.4 - 2 ns	2 - 4 ns
Bipolar	50 - 200 ps	200 - 400 ps
GaAs	20 - 100 ps	100 - 250 ps

Table K.4: Typical rise/fall times for IC technologies. Almost redrawn from [6] tab.6.3.

pure capacitance. The one-Gbps lines require either bipolar or GaAs drivers. Here the rise and fall times are short and lines may be long enough to require transmission line considerations. When transmission line considerations have to be made, recommended characteristic impedance for CMOS is $80 - 100\Omega$ and for bipolar systems $60 - 80\Omega$ ([24], [6] pp 288- 289).

K.8 Thermal considerations.

Many failure mechanisms increase with temperature. Thus removal of heat generated from circuits is important for proper function. In our design the most important failure mechanism is:
• Mechanical stress in material interfaces due to different expansion coefficients (important for flip-chip mounted circuits).
Other disadvantages of increased temperature is:
• Increased electron migration,
• Increased probability of oxide breakdown,
• Worsened hot-electron injection effects,
• Increased leakage currents in reversed-biased junctions and turned-off transistors,
• Acceleration of corrosion mechanisms, and
• Reduced CMOS switching speed.

Recommended temperatures are in the range $0 - 70^{O}C$ for commercial devices and in the range -55 to $125^{O}C$ for devices for military use.

Heat transfer takes place through conduction, convection and radiation, of which conduction and convection are dominating. Conduction means heat transfer through solids like packet or circuit legs. Convection means heat transfer to quiet or moving gas (air or Helium) or liquid (typically water).

Heat conduction is expressed in the following equation (Fig. [6] 3.2):

$$\theta = \frac{(T_j - T_a)}{P} \tag{K.6}$$

The θ is the thermal resistance, T_j the junction (circuit) temperature, T_a the ambient temperature, and P the power generated by the circuit.

Equation K.6 may also be expressed as:

$$T_j = \theta P + T_a \tag{K.7}$$

Here T_j may be the temperature of a water-cooled substrate or of the surrounding air. If the circuit generates a power P and θ is the thermal resistance to the substrate or air, the circuit will have a temperature T_j. By reducing the thermal resistance the temperature of the circuit decreases.

K.8.1 Thermal conduction.

Due to high pin number and to reduce inductance and capacitance, flip-chip mounting of naked circuits has been chosen. Properly mounting of naked circuits, wire bounded (active area up) or flip-chip bounded (active area down), may give lower thermal resistance between chip and substrate than encapsulated alternatives. With basic flip-chip mounting all heat conduction takes place through bonding bumps. This gives a thermal resistance 4 - 5 times larger ([74]) than for wire bound circuits where the entire back side of the circuit can be glued on the substrate.

Different strategies can be used to reduce the thermal resistance between flip-chip-mounted circuits and the substrate. One is to increase the number of bonding bumps. Since an increase of signal bumps would increase the capacity and power consumption, it has to be the number of power bumps which is increased.

More power bumps increase the capacitance and reduce the inductance, which both are advantages. A hole in the upper signal layer and the polyimid layer should allow the power bumps direct access to the upper power plane. The substrate power lines are surrounded by electrically insulating layers with a high thermal conductivity. To reduce the chip-substrate thermic resistance further, the power metal layers beneath the chips are made as large as possible. A third action to be taken is to use an "underfill", i.e. to fill the air space between circuit and substrate with an elastic, isolating material with high thermic conductivity. With all these strategies to reduce thermic resistance implemented, the resistance may be below $6.0^{O}C/W$.

A thermal resistance of $6.0^{O}C/W$ is an acceptable value securing good off-loading of heat to the substrate for the expected power values. If the heat drains of the substrate (i.e. cabinet and heat sinks) can absorb the added heat, and the added heat is not too unevenly distributed, we can expect a temperature below $85^{O}C$ on each circuit. This should be verified by thermal simulation software.

K.8.2 Thermal convection.

The backside (upside) of the naked flip-chips will transfer some heat to surrounding air by convection. The convection can be increased by gluing a small heat sink on the backside of the circuit. The subject has only been superficially studied, but some thoughts can be presented: Since the switch will not have a fan the heat sink "fingers" have to be vertical. The only sinks with vertical fingers found by the author are characterised in lengths of 3 inches and above. Length adjustment calculations done by the auhor indicate that the thermal resistance of a heat sink with length 1 cm is 2.5 times the resistance of a 3-inch sink. Thus it should be possible to find suitable heat sinks with a resistance below $45^{O}C/W$. The heat sink must have a low weight to contribute with as little mechanical stress on the bonding bumps as possible. There will be a crossover-point above which a heavy heat sink adds more failures due to mechanical stress than it reduces failures by lowering the chip temperature. This is a complicated function which is outside the scope of this thesis.

K.8.3 Stress due to different thermal expansion factors.

The bonding bumps will experience mechanical stress due to different thermic expansion factors on chip and substrate. We will in the following find an expression for this stress.

Definitions of abbrevations used in the following		
E_{Si}	Thermal expansion factor for Silicon	$2.5 \cdot 10^{-6} \cdot 1/^{O}C$
E_{SiC}	Thermal expansion factor for Silicon carbide	$3.7 \cdot 10^{-6} \cdot 1/^{O}C$
δx	Distance from chip center to corner	$0.75cm$
T_{Si}	Temperature of chip	
T_{SiC}	Temperature of substrate	
T_l	Common temperature at time of soldering	
X_{Si}	Change in corner-center distance on chip	
X_{SiC}	Change in corner-center distance on substrate	
ΔX	Difference in expansion between chip and substrate	$X_{SiC} - X_{Si}$
h	Height of soldier bumps	
S	Stress factor	$S = \Delta X/h$

X_{Si} is the change in distance between the position of the corner pad on the chip and the chip center. X_{SiC} is the change in distance between the position of the corner pad on the substrate and the point beneath the chip center. Both X_{Si} and X_{SiC} are given relative to the distance at T_l. Thus, when $T_{Si} = T_{SiC} = T_l$ we have $X_{Si} = X_{SiC} = 0$.

$$X_{Si} = E_{Si} \cdot \delta x \cdot (T_{Si} - T_l) = E_{Si} \cdot \delta x \cdot (\theta \cdot P + T_{SiC} - T_l) \tag{K.8}$$

$$X_{SiC} = E_{SiC} \cdot \delta x \cdot (T_{SiC} - T_l) \tag{K.9}$$

We define $\Delta X = X_{SiC} - X_{Si}$. By inserting $\theta = 6^{O}C/W$ and $P = 2W$ in the equations above we end up with the following expression:

$$\Delta X = (T_{SiC} - T_l) \cdot 9nm/^{O}C - 225nm \tag{K.10}$$

This equation expresses the difference in position of a corner pad on an integrated circuit relative to the corresponding point on the substrate. This will be the horizontal stress over a pad bump between the upper and lower end. The mechanical stress over a bump is expressed as:

$$S = \Delta X / h \tag{K.11}$$

K.8.4 Measured life times of power cycling tests.

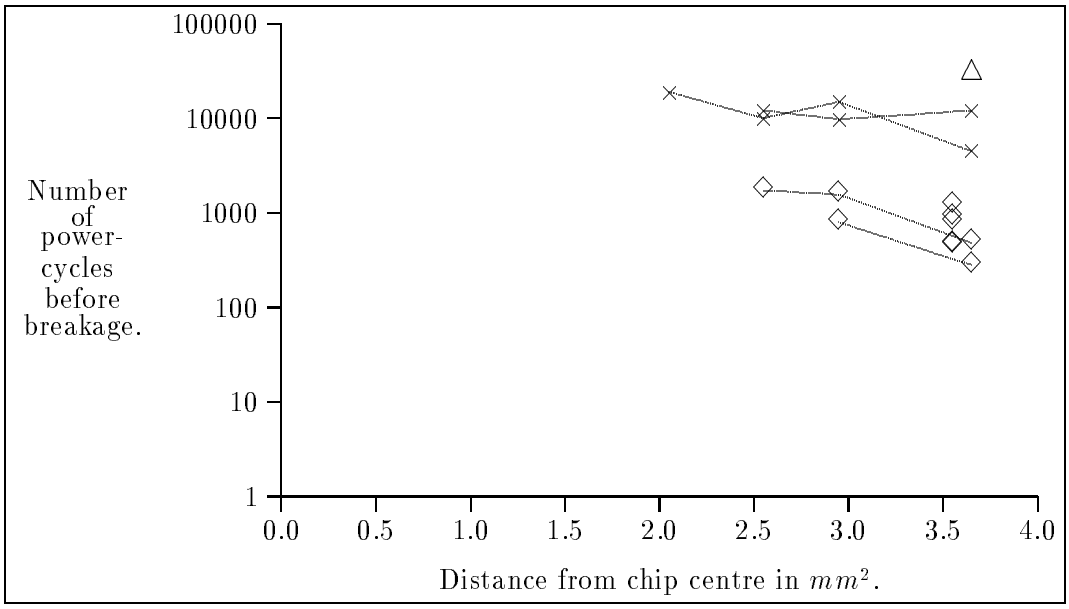


Figure K.5: The figure shows reliability measurements of some flip-chip mounted circuits. The measurements have been performed as an accelerated aging process where power has been switched on and off in cycles of 30 or 60 seconds. The measurements marked by ◇ (made by the author) has a cycle time of 30 seconds with a temperature difference of $110^{O}C$. *The single measurement marked by a △ has two differences to the previously mentioned: The circuit was mounted with flip-chip underfill and the measurement had to be terminated without breakage. At the time of the termination of the measurement the behaviour where similar to the initial behaviour. The power consumption of these circuits were between 15 and 27 W. Two circuits had a temperature difference between circuit and substrate of* $45 - 65^{O}C$. *The measurements of these circuits are marked by* $\times$. *The power consumption on these circuits where* $10 - 12W$. *The conclusions from the measurements were not unexpected: time before breakage decreases with increasing distance from chip center and increases with increasing temperature difference. Another important conclusion is that flip-chip underfill increases time to breakage by an order of ten or more.*

Conclusion to thermal consideration.

A switch circuit can be implemented as described, without resulting in damaging chip temperatures. The analyses also show that the main heat drain will be through conduction to the substrate. Heat sinks can be added on each circuit to reduce temperature further, although this possibility is not yet fully examined.

K.9 Conclusion.

The switch specifications are small physical size, high bandwidth and a design without fan or liquid cooling. The switches are intended to be placed around in offices. Thus the energy consumption and the heat amount generated has to be as small as possible. Reduction of power consumption can be obtained through the choice of optimal substrate material, wire routing, circuit mounting and circuit design. For substrate material, wire routing and circuit mounting the same choices support low power consumption and high clock rate. Bandwidth is the product of clock rate and bus width. The trade-off between clock rate and bus width described in this thesis is based on considerations explained in this appendix.

K.9.1 Total power consumption.

By adding the values found for the CSU, sixteen IPs and sixteen OPs connected by optimal wiring, the total power consumption with a $5V$ supply is $5W$. With reduced voltage supply and increased word length to support the same bandwidth, at $3.3V$ supply the consumption is $2.8W$. When the power supply is further reduced in the same way, the consumption is $2W$ at $2.5V$ and $1.25W$ at $1.5V$.

The total power consumption for 16 pairs of the commercially available optical modules from [80], is $73.6W$. Since this is more than fourteen times the power consumption of the remaining part of the switch it is clear that the optical modules are the main target for power reduction.

Appendix L:
Examples of VHDL code

jjswitch						
	jswitch					
		switch				
			cb83			
				cninp3		
					cbimrg	
					cc	
					dec3to8	
					muxmem	
						dff
					cbinpc	
				cbout3		
					cbocbo	
					cbocb	
					cboutmux	
					arbiter	
						vdd
						cboutarc
			opi			
				opdipene		
				opomig		
				gfifo8		
			ipi			
				efifo16		
				ipipre		
				ipc		

Table L.1: The table shows the top hierarchy for the VHDL code. The code is for a system of four switches.

L.1 The Input Port pipeline: "ipc"

The VHDL code is based on the layout described in appendix B. The description in appendix B is more detailed than the comments in the VHDL code. The naming of registers and bits is kept corresponding when possible.

```
-- VHDL Model Created from SGE Symbol ipc.sym -- Jan 23 06:27:45 1994

library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_components.all;

entity IPC is
      Port (     CLK : In    std_logic; -- Clock signal
                  IG : In    std_logic; -- Flow control  IfcI
                IPCI : In    std_logic_vector (9 downto 0);
                   -- Incoming 10-bit data bus
               RESET : In    std_logic;
              RETADR : In    std_logic_vector (3 downto 0);
                   -- The input channel number
                   -- (fixed for each input channel)
             TIMEOUT : In    std_logic;
                   -- Initiate a packet dismiss if passive for 50ms
                 BGI : In    std_logic;
                   -- Flow control Ofc0
               SHIFT : Out   std_logic; -- Shift signal
                   -- Shift one symbol out of main buffer
                IPCO : Out   std_logic_vector (4 downto 0) );
                   -- Outgoing 5-bit data bus
end IPC;

architecture BEHAVIORAL of IPC is
  signal A,B,C,D,E,F,G,H,I: std_logic_vector (8 downto 0);
  -- Pipeline registers with one bit indicating control or data symbol
  -- and eight bit of contents.
  signal AV,BV,BBOP,BEOP,BDF,CV,CBOP,CEOP,CDF:  std_logic;
  signal DV,DA4,DEOP,DDF,EV,EA4,EEOP,EDF:       std_logic;
  signal FV,FA4,FEOP,FDF,JC,N,GV,GA4,GEOP:      std_logic;
  signal HV,HA4,HEOP,R,M:                       std_logic;
  -- xV: register x has valid contents
  -- xBOP: register x contains a BOP symbol
  -- xEOP: register x contains a EOP symbol
  -- xDF: register x contains a DF symbol
  -- xA4: register x contains a A4 symbol
  -- Registers for state machine: JC, N, R and M
  signal SHIFTE,SHIFTF,LHWG,LHWBOP,LIWEOP,LIWH: std_logic;
  -- SHIFTE: Shift D into E
  -- SHIFTF: Shift E into F
```

```
  -- LHWG: Load H with G
  -- LHWBOP: Load H with BOP
  -- LIWEOP: Load I with EOP
  -- LIWH: Load I with H
  signal OBUS:              std_logic_vector (4 downto 0);
begin
 IPCO <= OBUS;
 SHIFT <= M; -- The M latch generates the global shift signal
 SHIFTE <= (M and (DV or ((not N) and ((not FA4) or (not EV)))));
 SHIFTF <= (M and ((not FA4) or (EV and DV)));

 LHWG   <= (((not R) and (not HA4)) or (R and (IG or (not HV))));
 LHWBOP <= ((not R) and HA4 and (not IG));
 LIWEOP <= (((not R) and IG) or (R and (HEOP or TimeOut)));
 LIWH   <= (((not R) and (not IG) and HA4)
          or (R and IG and HV and (not HEOP) and (not TimeOut)));
    M <= (Reset or
       (((not R) and (not HA4)) or (R and (IG or (Not HV))))) after 1 ns;

 InputPort:
 process(CLK)
 begin
  if CLK'Event and CLK = '1' then

    if (M = '1') then
      AV <= IPCI(9) after 1 ns;
      A(8 downto 0) <= IPCI(8 downto 0) after 1 ns;
    end if;


    -- B copies A,
    -- BOP, DF and EOP are identified
    -- EOP is high either because the register contains a EOP symbol
    --        or because the jump counter is zero.
    if (M = '1') then
      BV <= AV after 1 ns;
      B <= A after 1 ns;

      BBOP <= AV and (not A(8)) and A(7) and (not A(6)) and A(5) after 1 ns;
      BDF  <= AV and (not A(8)) and (not A(7)) and A(6)
                                                  and (not A(5)) after 1 ns;
      BEOP <= (AV and (not A(8)) and A(7) and (not A(6)) and (not A(5)))
              or (AV and A(8) and (not A(7)) and (not A(6)) and (not A(5))
                     and (not A(4)) and BBOP) after 1 ns;
    end if;


    -- If C is not BOP, C copies B,
    -- If C is BOP, C copies B and decrements the jump counter on the way
    -- (The logic verifies that B has valid contents before copy)
```

```
if (M = '1') then
  CV <= (BV or (CBOP and CV) ) after 1 ns;
  C(8) <= ((CBOP and (not BV) and C(8)) or ((not CBOP) and B(8))
                                    or (BV and B(8))) after 1 ns;
  C(7) <= (((not CBOP) and B(7)) or (CBOP and BV and
                ((B(7) and (B(6) or B(5) or B(4)))
         or ((not B(7)) and (not B(6)) and (not B(5)) and (not B(4)))))
                                    or (CBOP and (not BV) and C(7))) after 1 ns;
  C(6) <= (((not CBOP) and B(6)) or (CBOP and BV and
                ((B(6) and (B(5) or B(4)))
                 or (not B(6) and (not B(5)) and (not B(4)))))
                                    or (CBOP and (not BV) and C(6))) after 1 ns;
  C(5) <= (((not CBOP) and B(5)) or (CBOP and BV and
                ((B(5) and B(4)) or ((not B(5)) and (not B(4)))))
                 or (CBOP and (not BV) and C(5))) after 1 ns;
  C(4) <= (((not CBOP) and B(4)) or (CBOP and BV and (not B(4)))
                 or (CBOP and (not BV) and C(4))) after 1 ns;
  C(3) <= ((CBOP and (not BV) and C(3)) or ((not CBOP) and B(3))
                 or (BV and B(3))) after 1 ns;
  C(2) <= ((CBOP and (not BV) and C(2)) or ((not CBOP) and B(2))
                 or (BV and B(2))) after 1 ns;
  C(1) <= ((CBOP and (not BV) and C(1)) or ((not CBOP) and B(1))
                 or (BV and B(1))) after 1 ns;
  C(0) <= ((CBOP and (not BV) and C(0)) or ((not CBOP) and B(0))
                 or (BV and B(0))) after 1 ns;
  CBOP <= ((CBOP and (not BV)) or BBOP) after 1 ns;
  CEOP <= BEOP after 1 ns;
  CDF  <= BDF after 1 ns;
end if;
if (RESET = '1') then
  CV <= '0' after 1 ns;
  C(8) <= '0' after 1 ns;
  C(7) <= '0' after 1 ns;
  C(6) <= '0' after 1 ns;
  C(5) <= '0' after 1 ns;
  C(4) <= '0' after 1 ns;
  C(3) <= '0' after 1 ns;
  C(2) <= '0' after 1 ns;
  C(1) <= '0' after 1 ns;
  C(0) <= '0' after 1 ns;
  CBOP <= '0' after 1 ns;
  CEOP <= '0' after 1 ns;
  CDF <= '0' after 1 ns;
end if;


-- If C does not contain a BOP, D will copy C
-- If C contains a BOP,
--            D copies an a4 symbol and the first address nibble from B
-- (The logic verifies that B and C have valid contents before copying)
```

```
if (M = '1') then
  DV <= CV and ((not CBOP) or BV) after 1 ns;
  D(8) <= C(8) after 1 ns;
  D(7) <= C(7) and ((not CBOP) or BV) after 1 ns;
  D(6) <= ((not CBOP) and C(6)) or (CBOP and BV) after 1 ns;
  D(5) <= C(5) and (not CBOP) after 1 ns;
  D(4) <= ((not CBOP) and C(4)) after 1 ns;
  D(3) <= ((not CBOP) and C(3)) or (CBOP and BV and B(3)) after 1 ns;
  D(2) <= ((not CBOP) and C(2)) or (CBOP and BV and B(2)) after 1 ns;
  D(1) <= ((not CBOP) and C(1)) or (CBOP and BV and B(1)) after 1 ns;
  D(0) <= ((not CBOP) and C(0)) or (CBOP and BV and B(0)) after 1 ns;
  DA4  <= CBOP and BV;
  DEOP <= CEOP;
  DDF  <= CDF;
end if;


-- The purpose of E and F is to manipulate the nibbles into correct
-- order and correct positions. This is explained with figures in
-- appendix B.
if (SHIFTE = '1') then
  E    <= D after 1 ns;
  EV   <= DV after 1 ns;
  EA4  <= DA4 after 1 ns;
  EEOP <= DEOP after 1 ns;
  EDF  <= DDF after 1 ns;
end if;
if (RESET = '1') then
  EV <= '0' after 1 ns;
end if;

if (SHIFTF = '1') then
  F(7) <= (E(3) and N and DV) or (E(7) and (not N)) after 1 ns;
  F(6) <= (E(2) and N and DV) or (E(6) and (not N)) after 1 ns;
  F(5) <= (E(1) and N and DV) or (E(5) and (not N)) after 1 ns;
  F(4) <= (E(0) and N and DV) or (E(4) and (not N)) after 1 ns;
  F(3) <= (E(3) and (not JC) and (not N)) or (RETADR(3) and DDF and N)
         or (D(7) and (JC or (DV and N and (not DDF)))) after 1 ns;
  F(2) <= (E(2) and (not JC) and (not N)) or (RETADR(2) and DDF and N)
         or (D(6) and (JC or (DV and N and (not DDF)))) after 1 ns;
  F(1) <= (E(1) and (not JC) and (not N)) or (RETADR(1) and DDF and N)
         or (D(5) and (JC or (DV and N and (not DDF)))) after 1 ns;
  F(0) <= (E(0) and (not JC) and (not N)) or (RETADR(0) and DDF and N)
         or (D(4) and (JC or (DV and N and (not DDF)))) after 1 ns;
  FV <= EV and ((not N) or DV) after 1 ns;
  F(8) <= E(8) and ((not N) or DV) after 1 ns;
  FA4 <= EA4 and ((not N) or DV) after 1 ns;
  FEOP <= EEOP and ((not N) or DV) after 1 ns;
  FDF <= EDF and ((not N) or DV) after 1 ns;
```

```
      N <= JC or (N and (not DDF)) after 1 ns;
      JC <= EA4 after 1 ns;
    end if;
    if (RESET = '1') then
      N  <= '0' after 1 ns;
      JC <= '0' after 1 ns;
      FA4<= '0' after 1 ns;
      FV <= '0' after 1 ns;
    end if;


    -- G copies F except
    --            when F (contains a4) and (D or E has not valid
    --            contents). If so, G copies idle symbols.
    if (M = '1') then
      GV   <= FV   and ((not FA4) or (EV and DV)) after 1 ns;
      G(8) <= F(8) and ((not FA4) or (EV and DV)) after 1 ns;
      G(7) <= F(7) and ((not FA4) or (EV and DV)) after 1 ns;
      G(6) <= F(6) and ((not FA4) or (EV and DV)) after 1 ns;
      G(5) <= F(5) and ((not FA4) or (EV and DV)) after 1 ns;
      G(4) <= F(4) and ((not FA4) or (EV and DV)) after 1 ns;
      G(3) <= F(3) and ((not FA4) or (EV and DV)) after 1 ns;
      G(2) <= F(2) and ((not FA4) or (EV and DV)) after 1 ns;
      G(1) <= F(1) and ((not FA4) or (EV and DV)) after 1 ns;
      G(0) <= F(0) and ((not FA4) or (EV and DV)) after 1 ns;
      GA4  <= FA4  and ((not FA4) or (EV and DV)) after 1 ns;
      GEOP <= FEOP and ((not FA4) or (EV and DV)) after 1 ns;
    end if;


    -- The purpose of the H register is to regenerate the BOP
    -- which were overwritten. After a4 has been transferred to
    -- the CSU, a BOP signal overwrites the a4 signal. After the BOP
    -- signal has been forwarded the contents of G is copied to H.
--    LHWG <= (((not R) and (not HA4)) or (R and (IG or (not HV))));
--    LHWBOP <= ((not R) and HA4 and (not IG));
    if (M = '1' and Reset = '0') then -- Load H with G
      H <= G after 1 ns;
      HV <= GV after 1 ns;
      HA4 <= GA4 after 1 ns;
      HEOP <= GEOP after 1 ns;
    elsif (R = '0' and HA4 = '1' and IG = '0') then -- Load H with bop
      HV <= '1' after 1 ns;
      HA4 <= '1' after 1 ns;
      HEOP <= '0' after 1 ns;
      H <= "010100000" after 1 ns;
    end if;
    if (RESET = '1') then
      HV  <= '0' after 1 ns;
      HA4 <= '0' after 1 ns;
```

```
    end if;


    R <= (not Reset) and
      (((not R) and HA4 and (not IG)) or (R and (not HEOP) and (not TimeOut)))
      after 1 ns;


    -- The CSU either receives an EOP or an IDLE symbol or
    -- the contents of the H register.
    if ((R = '0' and IG = '1')
      or (R = '1' and (HEOP = '1' or TimeOut = '1'))) then   -- EOP
            I <= "010000000" after 1 ns;
            OBUS(4 downto 0) <= "01000" after 1 ns;
    elsif ((R = '0' and IG = '0' and HA4 = '1')
      or (R = '1' and IG = '1' and HV = '1'
        and HEOP = '0' and TimeOut = '0')) then               -- ORD
            I <= H after 1 ns;
            OBUS(4 downto 0) <= H(8 downto 4) after 1 ns;
    else                                                      -- IDLE
            I <= "000000000" after 1 ns;
            OBUS(4 downto 0) <= "00000" after 1 ns;
    end if;
  end if;

  if CLK'Event and CLK = '0' then
    OBUS(3 downto 0) <= I(3 downto 0) after 1 ns;
    OBUS(4)      <= BGI after 1 ns;
  end if;
 end process;
end BEHAVIORAL;

configuration CFG_IPC_BEHAVIORAL of IPC is
   for BEHAVIORAL

   end for;

end CFG_IPC_BEHAVIORAL;
```

L.2 Synchrouns fifo: "gfifo8"

This is a high-level VHDL model for the synchronous FIFO in appendix G.1. The model gives an equal external logical behaviour. The internal description is simplified and few similarities can be found.

```
-- VHDL Model Created from SGE Symbol gfifo8.sym -- Jan 28 19:35:44 1994

library IEEE;
   use IEEE.std_logic_1164.all;
```

```
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_components.all;

entity GFIFO8 is
      Port (     CLK : In    std_logic;
                 INP : In    std_logic_vector (9 downto 0);
               RESET : In    std_logic;
               SHIFT : In    std_logic; -- Shift next word to FIFO output
                 FAF : Out   std_logic; -- FIFO Almost full
                FULL : Out   std_logic; -- FIFO full
                OUTP : Out   std_logic_vector (9 downto 0) );
end GFIFO8;

architecture BEHAVIORAL of GFIFO8 is

  constant S         : INTEGER := 8;  -- NR OF WORDS IN FIFO
  constant W         : INTEGER := 10; -- WORD WIDTH  ********
  signal   R         : std_logic_vector ((S * W) downto 0);
                       -- FIFO in one-dimensional vector format
                       -- In the VHDL code the FIFO is organised as a
                       -- ring buffer
  signal   OC        : std_logic_vector ((W-1) downto 0);
                       -- FIFO word at output of FIFO

   begin
  OUTP <= OC;

  RAM_SHIFT:
  process(CLK)

   constant FAFLIMIT  : INTEGER := 6;
                        -- The number of words for the FIFO limit
   variable ReadAdr   : INTEGER range 0 to (S+1);
   variable NrWords   : INTEGER range 0 to (S+1);
   variable J         : INTEGER;
   variable NewWrite  : INTEGER range 0 to 1;
  begin

if CLK'Event and CLK = '1' then

   -- Reset section
   if (Reset = '1') then
     ReadAdr := 0;
     NrWords := 0;
     FULL <= '0';
   end if;


   -- Fifo almost full
```

```
if (NrWords > FAFLIMIT) then
  FAF <= '1';
else
  FAF <= '0';
end if;

-- Write section
if (NrWords = S) then -- FIFO is full
  FULL <= '1';
  NewWrite := 0;
elsif (ReadAdr + NrWords) > (S - 1) then
  -- Write pointer has a lower value than read pointer
  for J in 0 to (W - 1) loop -- Write new word to FIFO
     R((ReadAdr + NrWords - S)*W + J) <= INP(J) after 1 ns;
  end loop;
  if (INP(W-1) = '1') then -- Valid contents
    NrWords := NrWords + 1;
    NewWrite := 1;
  else -- Empty contents
    NewWrite := 0;
  end if;
  FULL <= '0';
else
  -- Write pointer has a higher value than read pointer
  for J in 0 to (W - 1) loop -- Write new word to FIFO
     R((ReadAdr + NrWords)*W + J) <= INP(J) after 1 ns;
  end loop;
  if INP(W-1) = '1' then -- Valid contents in input word
    NrWords := NrWords + 1;
    NewWrite := 1;
  else -- Invalid contents in input word
    NewWrite := 0;
  end if;
  FULL <= '0';
end if;


-- Read section
if ((NrWords > 1) and SHIFT = '1')
   or (SHIFT = '1' and NewWrite = 0 and NrWords = 1) then
  -- The condition that the oldest word can be read, is that
  -- it is valid, and at least one clock period old
  ReadAdr := ReadAdr + 1;
  if ReadAdr > (S-1) then
  -- Correct Readpointer if outside FIFO size
     ReadAdr := ReadAdr - S;
  end if;
  NrWords := NrWords - 1;
end if;
```

```
   if ((NrWords > 1) or (NrWords = 1 and NewWrite = 0)) then
       OC((W-1) downto 0) <= -- Output word is read from FIFO
                       R(((ReadAdr*W)+W-1) downto (ReadAdr*W)) after 1 ns;
   elsif (NrWords = 1) then -- Output word is read from FIFO input
       OC((W-1) downto 0) <= INP((W-1) downto 0) after 1 ns;
   else -- FIFO is empty
       for J in 0 to (W-1) loop
         OC(J) <= '0' after 1 ns;
       end loop;
   end if;
end if;
end process;


end BEHAVIORAL;

configuration CFG_GFIFO8_BEHAVIORAL of GFIFO8 is
   for BEHAVIORAL

   end for;

end CFG_GFIFO8_BEHAVIORAL;
```

L.3 The elastic buffer: "efifo16"

The following VHDL code is for a FIFO with asynchronous input and output side. The FIFO is built around the two-port RAM. The contents of the write pointer are transferred from the write side to the read side. At the read side, the copied write pointer values give an upper limit to the amount of data which can be read. The concept is described in chapter *18 Implementation of the elastic FIFO*

```
-- VHDL Model Created from SGE Symbol efifo16.sym -- Jan 26 18:03:46 1994

library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_misc.all;
   use IEEE.std_logic_arith.all;
   use IEEE.std_logic_components.all;

entity EFIFO16 is
      Port (    CLKR : In    std_logic; -- Read Clock, Clock on receiving side
                CLKW : In    std_logic; -- Write Clock, Clock on transmitting side
                 INP : In    std_logic_vector (9 downto 0);
               RESET : In    std_logic;
               SHIFT : In    std_logic; -- Data at output used, shift next to front
                 FAF : Out   std_logic; -- FIFO Almost Full
                OUTP : Out   std_logic_vector (9 downto 0) ); -- Word at FIFO output
end EFIFO16;
```

```
architecture BEHAVIORAL of EFIFO16 is
  constant S         : INTEGER := 16;  -- Nr of words in FIFO ********
  constant W         : INTEGER := 10; -- Word width  ********
  signal   R         : std_logic_vector ((S * W) downto 0); -- Buffer on vector format
                       -- The FIFO is organised as a ring FIFO in the VHDL code
                       -- This is also how it is thought implemented in the layout
  signal   OC        : std_logic_vector ((W-1) downto 0); -- Word at FIFO output
  signal   T         : std_logic; -- Token used between transmitting and receiving
                                  -- part for exchange of write pointer

   begin
  OUTP <= OC; -- Block output equal to FIFO output

  RAM_SHIFT:
  process(CLKR,CLKW)
   constant FAFLIMIT    : INTEGER := 12;-- Has to be corrected ********
-- FAFLIMIT depends on the bandwidth - flow control roundtrip product
-- FAFLIMIT have to be between 1 and S
   variable R_ReadAdr   : INTEGER range 0 to (S+1); -- Read pointer
   variable W_WriteAdr  : INTEGER; -- Write pointer
   variable WR_WriteAdr : INTEGER range 0 to (S+1);
                                   -- Copy of write pointer on write side
                                   -- This copy will be read by the reading side
   variable R_WriteAdr1 : INTEGER; -- First copy of Write pointer on reading side
   variable R_WriteAdr2 : INTEGER; -- Second copy of Write pointer on reading side
   variable HandShake   : INTEGER;
   variable J           : INTEGER;
   variable Carry       : INTEGER range 0 to 1;
   variable NrWords     : INTEGER;
   variable validout    : INTEGER range 0 to 1;
  begin

-- ******************************************************
-- *           Write / (Transmitting) side of FIFO      *
-- ******************************************************
if CLKW'Event and CLKW = '1' then

   -- Reset section
   if (Reset = '1') then
     W_WriteAdr := 0;
     WR_WriteAdr:= 0;
     T <= '0';
   end if;

   -- Write section
     for J in 0 to (W - 1) loop -- Write incoming word to RAM
        R(W_WriteAdr*W + J) <= INP(J) after 1 ns;
     end loop;

     if (INP(W-1) = '1') then -- If the word read was valid,
```

```
                            -- increment the writepointer
      W_WriteAdr := W_WriteAdr + 1;
      if (W_WriteAdr = S) then W_WriteAdr := 0; end if;
    end if;

    if (T = '0') then -- If "token" is on the write side
      WR_WriteAdr:= W_WriteAdr;
               -- Copy write pointer to transfer register
      T <= '1';
    end if;
end if;
-- ******************************************************
-- *           Read / (Receiving) side of FIFO          *
-- ******************************************************
if CLKR'Event and CLKR = '1' then

   -- Reset section
   if (Reset = '1') then
     R_ReadAdr  := 0;
     WR_WriteAdr:= 0;
     R_WriteAdr1:= 0;
     R_WriteAdr2:= 0;
     HandShake:= 0;
     Carry :=  0;
     T <= '0';
     validout:= 0;
   end if;

   -- Read section

   -- Handshake section
   if (T = '1') then HandShake:= HandShake + 1;end if;
   -- Count how long "T" has been high

   if (HandShake = 2) then
     R_WriteAdr1 := WR_WriteAdr;
     -- Load into first Write pointer register
     if R_WriteAdr1 < R_WriteAdr2 then Carry := 1; end if;
   end if;
   if (HandShake = 3) then
     -- Load into second Write pointer register
     R_WriteAdr2 := R_WriteAdr1;
     T <= '0'; -- Return token to Writing half of FIFO
     HandShake:=0;
   end if;

   -- Calculate the number of words in FIFO
   if (Carry = 1) then NrWords := S + R_WriteAdr1 - R_ReadAdr;
   else                NrWords := R_WriteAdr1 - R_ReadAdr;
   end if;
```

```
   -- Copy to output
   if (NrWords = 0) then -- FIFO is Empty: Idle symbol to output
     for J in 0 to (W-1) loop
       OC(J) <= '0' after 1 ns;
     end loop;
     validout := 0;
   else -- FIFO has valid contents
      if ((NrWords > 0) and (shift = '1') and (validout = 1)) then
         -- Data at read pointer value forwarded to FIFO output
         -- Read pointer increments
         NrWords := NrWords - 1;
         R_ReadAdr := R_ReadAdr + 1;

         if (R_ReadAdr = S) then
            R_ReadAdr := 0;
            Carry := 0;
         end if;

         if (NrWords > 0) then
            OC((W-1) downto 0) <=
            R(((R_ReadAdr*W)+W-1) downto (R_ReadAdr*W)) after 1 ns;
            validout := 1;
         else

           for J in 0 to (W-1) loop
             OC(J) <= '0' after 1 ns;
           end loop;
           validout := 0;
         end if;
      else
         -- Data at read pointer value forwarded to FIFO output
         -- Read pointer not incremented
         OC((W-1) downto 0) <=
                  R(((R_ReadAdr*W)+W-1) downto (R_ReadAdr*W)) after 1 ns;
         validout := 1;
      end if;
   end if;

   -- FAF = FIFO Almost Full
   if ((S - NrWords) < FAFLIMIT) then
      FAF <= '0';
   else
      FAF <= '1';
   end if;
end if;
-- ***************************************************
end process;

end BEHAVIORAL;
```

```
configuration CFG_EFIFO16_BEHAVIORAL of EFIFO16 is
   for BEHAVIORAL

   end for;

end CFG_EFIFO16_BEHAVIORAL;
```

April 1997

Full custom implementation of a high performance input buffered switch architecture

Joar Martin Østby

Dr Scient thesis

Research Report 242

Department of Informatics

Faculty of Mathematics and Natural Sciences
UNIVERSITY OF OSLO